
Einsteigen - Verstehen - Beherrschen
DM 3,80 öS 30 sfr 3,80
computer
kurs
FORTH-Kurs
Eine Zusammenstellung
aus mehreren Heften
dieses Computer-
magazins
Ein wöchentliches
Sammelwerk

. . . und so FORTH

FORTH ist eine Programmiersprache, die ursprünglich zur Steuerung eines Radio-Teleskops entwickelt wurde. Unsere Einführung betrachtet zwei wesentliche Eigenschaften dieser Sprache: Interaktivität und Erweiterbarkeit.

FORTH wurde Ende der sechziger Jahre von Charles Moore speziell zur Steuerung wissenschaftlicher Instrumente entwickelt, die eine sehr genaue Justierung benötigen. Die Leistungsfähigkeit dieser Sprache geht weit über reine Steuerungsanwendungen hinaus. Zur Einführung in die grundlegenden Eigenschaften von FORTH ist es jedoch sinnvoll, sich einmal zu überlegen, wie beispielsweise ein Teleskop durch FORTH gesteuert würde.

Es wäre doch ganz nett, wenn wir in der Lage wären, am Computerterminal in der Sternwarte einfach einzutippen:

30 GRAD ELEVATION

und die Einstellung des Teleskops würde sich sofort genau um diesen Wert ändern. Es wäre darüber hinaus sinnvoll, häufig verwendete Befehlssequenzen zusammenfassen und abspeichern zu können, um sich unnötige Tipparbeit zu ersparen. Wenn zum Beispiel die Einstellung des Teleskops 90 Grad Elevation und 0 Grad Azimut betragen würde, könnte man einen Befehl mit dem Namen „PARK" definieren, nämlich als

0 GRAD AZIMUT 90 GRAD ELEVATION

Der Jupiter Ace war ein mutiger Versuch, eine Maschine auf den Markt zu bringen, die standardmäßig mit FORTH ausgestattet war. Leider konnte sich der Jupiter Ace wegen seiner fehlenden Farbgrafik, seines zu kleinen Arbeitsspeichers und des zu geringen Bekanntheitsgrads von FORTH nicht auf dem Heimcomputermarkt durchsetzen.
Zusammen mit der hervorragenden Originalliteratur bietet er jedoch einen ausgezeichneten Einstieg in die Sprache.

Insgesamt benötigen wir also

- elementare Teleskop-Befehle,
- eine Möglichkeit, diese Befehle zu komplexeren Befehlen (zu einem Teleskop-Programm) zusammenfassen zu können.

Aus einem solchen System würde am Ende eine eigene, komplette Teleskop-Programmiersprache entstehen, die über besondere Eigenschaften verfügt.

Stellen Sie sich zunächst einmal vor, Sie müßten diese Aufgabe in BASIC lösen. Zwar lassen sich hier die fest eingebauten Befehle wie PRINT oder RUN direkt über die Tastatur eingeben, problematisch wird es aber mit den neu definierten Befehlen.

ELEVATION und AZIMUT

Nehmen wir an, sie haben Ihre Befehlserweiterungen ELEVATION und AZIMUT als Unterprogramme in den Zeilen 1000 bzw. 1100 abgelegt. Der Befehl PARK würde so aussehen:

```
1200 REM PARK
1205 LET GRAD = 0
1210 GOSUB 1000
1215 LET GRAD = 90
1220 GOSUB 1100
1225 RETURN
```

Eine andere Möglichkeit wäre, zunächst eine entsprechende Variable zu definieren:

```
LET PARK = 1200
```

und dann die Befehlssequenz aufzurufen, indem Sie die Variable im GOSUB angeben:

```
GOSUB PARK
```

Am einfachsten ist es jedoch, wenn Sie den Unterprogrammteil als eine Prozedur definieren (dies ist zum Beispiel im Acorn-BASIC oder in zahlreichen BASIC-Erweiterungen möglich). Der Aufruf würde dann etwa in der Form

```
PROCPARK
```

erfolgen. Eine Alternative wäre es, einen Interpreter zu schreiben, der die Befehle entgegennimmt und ausführt, wie beispielsweise

```
100 INPUT C$
110 IF C$ = "PARK" THEN GOSUB 1200 :
  GOTO 100
```

Allerdings ist es nun nicht mehr möglich, die eingebauten BASIC-Befehle direkt über die Tastatur einzugeben.

Aus dem Vorangegangenen war zu ersehen, daß wir zwei Forderungen an unsere Teleskop-Sprache stellen müssen:

● Erstens muß sie interaktiv sein. Das heißt, sie muß in der Lage sein, die Befehle direkt auszuführen, nachdem sie eingetippt wurden. Diese Eigenschaft besitzen Sprachen wie BASIC, LOGO, APL und PROLOG, nicht aber Sprachen wie PASCAL, ALGOL, FORTRAN oder COBOL. Auch die Kommandointerpreter von Betriebssystemen wie CP/M oder UNIX besitzen diese Eigenschaft.

● Zweitens muß sie erweiterbar sein. Das Eingabeformat für eingebaute Befehle muß dasselbe sein wie für selbstdefinierte Befehle. Nachdem wir einige Erweiterungen durchgeführt haben, können Sie so arbeiten, als hätten Sie es mit einer erweiterten, leistungsfähigeren Sprache, und nicht mit der um einige zusätzliche Routinen erweiterte Originalsprache zu tun. Das ist in LOGO oder den erwähnten Betriebssystemen möglich, aber nicht in den meisten anderen Sprachen.

Das Konzept der Erweiterbarkeit ist allerdings noch wesentlich leistungsfähiger. Wenn uns jemand eine Teleskop-Steuerungssprache verkauft und wir sie um einige nützliche Routinen für unser eigenes Teleskop ergänzen, so haben wir die ursprüngliche „allgemein einsetzbare Teleskop-Steuerungssprache" zu einer „persönlichen Teleskop-Steuerungssprache" erweitert.

Soll aber eine Erweiterung in vollem Umfang durchgeführt werden, so sollten wir mit einer „allgemein einsetzbaren erweiterbaren Computersprache" beginnen und diese zu einer „allgemein einsetzbaren erweiterbaren Teleskop-Steuerungssprache" erweitern können. Auf diese Art können nicht nur Teleskope, sondern auch Roboter, wissenschaftliche Experimentiergeräte, Produktionsanlagen, – kurz alles das, was sich an einen Computer anschließen läßt, gesteuert und kontrolliert werden.

FORTH-Dialekte

Von FORTH gibt es im wesentlichen drei Dialekte. Da wäre einmal das FIG (Forth Interest Group) FORTH, FORTH-79 und FORTH-83. Die meisten FORTH-Versionen beziehen sich auf einen dieser drei „Standards".

FORTH-79 und FORTH-83 sind aufeinanderfolgende Standards und definieren nur den grundlegenden Befehlssatz. Jede einzelne FORTH-Version wird sicher einen größeren Befehlssatz zur Verfügung stellen, der spezielle Eigenschaften des Computers (wie etwa Sound oder Grafik) unterstützt. Wenn Sie ein Programm schreiben, das sich strikt an den FORTH-83-Standard hält, so sollte es auf allen FORTH-83-Implementationen laufen. FORTH-83 ist der aktuelle Standard, und wir werden uns daran halten. FORTH-79 bietet zusätzlich eine Reihe von Befehlen, die die Implementierung des jeweiligen FORTH-Systems unterstützen. Viele FORTH-79- und FORTH-83-Implementationen gehen auf FIG-Forth zurück und werden erweitert, um dem jeweiligen Standard zu entsprechen.

Das Teleskop und die Turtle

Es lohnt sich, einen Vergleich zwischen LOGO und FORTH durchzuführen, da sich die beiden Sprachen sehr ähnlich sind. Beide wurden ursprünglich zur Kontrolle und Steuerung eines physikalischen Objekts entwickelt, eines Teleskops bzw. einer Schildkröte.

Beide Sprachen haben zwei wesentliche übereinstimmende Arbeitsprinzipien: Eine auszuführende Routine wird einfach durch ihren Namen aufgerufen, und neue Routinen werden auf bereits existierenden Routinen aufgebaut. Der eigentliche Unterschied liegt eher in den verschiedenen Erwartungen ihrer Benutzer. FORTH-Anwender erwarten eine hohe Arbeitsgeschwindigkeit (deshalb ist FORTH nicht nur interaktiv und erweiterbar, sondern auch sehr effizient) und nehmen dafür einige Unbequemlichkeiten in Kauf. LOGO dagegen ist konzipiert worden, um Kindern das Programmieren beizubringen. Es ist einfach zu erlernen und erlaubt daher keine Unbequemlichkeiten. Die Folge ist, daß LOGO wesentlich langsamer arbeitet. FORTH hat standardmäßig keine „Turtle-Grafik" eingebaut, verfügt aber über Möglichkeiten, den Grafik-Bildschirm zu steuern. Diese elementaren Befehle können Sie dazu verwenden, die LOGO-Befehle „FORWARD, RIGHT" usw. zu definieren. Vergleichen Sie die beiden Möglichkeiten, die Turtle zu steuern:

LOGO	FORTH
FORWARD 100	100 FORWARD

Bei diesem Beispiel könnte man den Eindruck gewinnen, FORTH würde sozusagen rückwärts arbeiten. Parameter in einer FORTH-Routine müssen zuerst berechnet und deshalb vor dem Namen der Routine eingegeben werden. Dieses Prinzip könnte man als „Rezeptbuch-Technik" beschreiben. Zuerst werden die Zutaten in bestimmten Mengen zusammengegeben und dann gegart.

Weitere Übereinstimmungen mit LOGO finden Sie bei der Definition neuer Routinen:

```
TO . . . END            : . . . ;
```

und bei der Schleifenkonstruktion:

```
REPEAT 4 [ . . . .]     40 DO . . . . LOOP
```

Hier ein Beispiel, wie wir beides zu einer Routine kombinieren können, die ein Quadrat auf dem Bildschirm zeichnet.

```
TO QUADRAT SEITE      : QUADRAT
REPEAT 4 [              40 DO
   FORWARD SEITE           DUP FORWARD
   RIGHT 90                   RIGHT
   ]                    LOOP
                        DROP
END                   ;
QUADRAT 100             100 QUADRAT
```

DUP und DROP sind „Stackmanipulationen", die den Routinen ermöglichen, auf die Parameter der QUADRAT-Routine zuzugreifen. DUP kopiert die Zahl für die FORWARD-Routine, DROP löscht die Zahl.



Es existiert mittlerweile für fast jeden Heimcomputer eine FORTH-Implementation. FORTH lebt wie wohl kaum eine andere Computersprache von ihren Anwendern. Die Verbreitung von FORTH und die Unterstützung seiner Anwender ist auch das Ziel der als Verein arbeitenden FORTH Interest Group. In Deutschland hat diese Aufgabe die FORTH-Gesellschaft übernommen: FORTH GESELLSCHAFT DEUTSCHLAND e. V., Schanzenstr. 27, 2000 Hamburg 6.

Geflügelte Worte

Programme in FORTH werden aus „Wörtern" aufgebaut, die der Programmierer selbst definieren kann. Die Wörter sind als Symbolgruppen definiert, die durch Leerzeichen voneinander getrennt werden. Wir untersuchen das „Vokabular" der Sprache, sehen uns an, wie FORTH Befehle, Konstanten und Variablen definiert.

Die Programmiersprache FORTH legt für ihre Befehle ein „Vokabular" an, das wie ein Wörterbuch eine Anzahl Wörter mit den zugehörigen Definitionen enthält. Die Wörter sind nicht alphabetisch gespeichert, sondern in der Reihenfolge, in der sie definiet wurden. Beim Aufruf eines Wortes teilt die Definition dem Computer mit, welche Abläufe er ausführen soll. Wörter können dabei entweder direkt eingegeben werden oder Bestandteil der Definition eines anderen Wortes sein.

FORTH interpretiert alles – Programme wie Daten – als Wörter. Die einzige Ausnahme sind Zahlen. Wörter lassen sich als Routinen (mit den Parametern Doppelpunkt und Semikolon), Variablen oder Konstanten definieren. Die Definitionen müssen aber in jedem Fall im Vokabular eingetragen sein.

Auch die ins System eingebauten Befehle sind Wörter. So stellt beispielsweise der Befehl WORDS (manche Systeme verwenden VLIST – die Kurzform für VokabularLISTe) eine Wörterliste des Vokabulars dar, in der der Befehl WORDS selbst als Eintrag auftaucht. Befehle von erweiterten FORTH-Versionen sind ebenfalls Wörter – Beispiele dafür sind GRAD und ELEVATION des Telescope-FORTH und FORWARD und RIGHT des Turtle-FORTH.

Vom Anwender definierte Befehle beginnen immer mit einem Doppelpunkt und enden mit dem Semikolon:

Die Zahlen

Vor der Definition

```
0 CONSTANT 0
```

war 0 eine Zahl, die durch Regel 2 erkannt wurde. Nach der Definition ist sie ein Vokabulareintrag, der von Regel 1 erkannt wird, bevor Regel 2 überhaupt zur Anwendung kommt. Da die Zahlen 0, 1 und 2 oft vorkommen, werden sie von figFORTH im Vokabular definiert.
Es ist aber auch die Definition

```
12 CONSTANT 13
```

möglich, bei der Sie jedesmal, wenn Sie 13 eingeben, die Zahl 12 erhalten. (Durch die Eingabe von 013 wird jedoch der korrekte Wert [13] ausgegeben.)

```
:PARK 0 GRAD AZIMUT 90 GRAD ELEVATION;
```

Dabei kommt zuerst der (:), dann das definierte Wort, danach die Definition und schließlich ein (;). Die einzelnen Komponenten der Definitionen müssen durch Leerzeichen getrennt werden. Definitionen lassen sich aber auch auf mehrere Zeilen aufteilen. Sie werden dadurch übersichtlicher:

```
:PARK
0 GRAD AZIMUT
90 GRAD ELEVATION
;
```

Selbst die Programme werden als Wörter bezeichnet. Wenn Definitionen aus einer einzigen langen Befehlskette bestehen – beispielsweise mit dem Namen RUN – dann können Sie RUN als Programm ansehen, das andere Wörter als Subroutinen verwendet. Dabei können durchaus weitere Befehle/Programme im Vokabular eingetragen sein. Subroutinen sind Wörter, die in anderen Definitionen eingesetzt werden.

FORTH unterscheidet nicht zwischen Befehlen, Programmen und Subroutinen. Alle sind mit (:) und (;) definiert und können direkt (über die Tastatur) oder indirekt (von einer anderen Definition) aufgerufen werden. Da diesen Definitionen ein Doppelpunkt (englisch: Colon) voransteht, werden sie „Colon-Definitionen" genannt.

Variablen sind ebenfalls Wörter. Die Variable LAENGE wird zum Beispiel so definiert:

```
VARIABLE LAENGE
```

Bei dieser Definition wird dem Wort LAENGE im Vokabular zusätzlicher Speicherplatz für einen Wert zur Verfügung gestellt. Variablen müssen deklariert werden, bevor sie eingesetzt werden können, da sie ohne Vokabulareintrag nicht als Variablen erkannt werden. Die Zeichen @ („holen") und ! („speichern") werden in Verbindung mit Variablen eingesetzt.

```
LAENGE @
```

ergibt den Wert von LAENGE und

```
26 LAENGE !
```

setzt den Wert von LAENGE auf 26 (in BASIC identisch mit: LET LAENGE = 26).

Das @ ist unbedingt notwendig, wenn Sie den Wert einer Variablen abfragen möchten. Ohne das @ erhalten Sie lediglich die Adresse der Variablen.

Konstante sind ebenfalls Wörter. Wenn ein Wert häufig in einem bestimmten Zusammenhang verwendet wird, sollten Sie ihn als Konstante definieren, zum Beispiel:

```
66 CONSTANT KLICKKLICK
```

Das Wort KLICKKLICK hat nun die Bedeutung 66. Sie können natürlich auch eine Variable verwenden, doch lassen sich Konstante ohne @ einsetzen. Allerdings funktioniert der Speicherbefehl (!) nicht mit Konstanten.

Deklarationen mit VARIABLE und CONSTANT dürfen nicht innerhalb einer Colon-Definition eingesetzt werden. Der Grund dafür liegt in der Art, wie Definitionen im Vokabular gespeichert sind. Zwar wäre es praktisch,

```
:INITIALISIERE
VARIABLE PUNKTE 0 PUNKTE !
VARIABLE SPIELE 0 SPIELE !
```

definieren zu können, doch es funktioniet leider nicht. Wie in BASIC sind alle Variablen „global". Lokale Variable wie in PASCAL gibt es in FORTH nicht.

FORTH arbeitet mit +, –, *, / und vielen anderen arithmetischen Operatoren, die wie alle Wörter mit Leerzeichen von anderen getrennt sein müssen. Die Argumente werden jeweils vorangestellt. Statt 2 + 3 wird

```
2 3 +
```

geschrieben. In der nächsten Folge werden wir darauf noch ausführlicher eingehen.

Der Doppelpunkt, VARIABLE und CONSTANT werden „Definitionswörter" genannt, da sie Definitionen im Vokabular anlegen und somit eine Sonderfunktion ausführen. Sie bleiben aber Wörter des Vokabulars. Es ist möglich, neue Definitionswörter anzulegen.

Unter den Symbolen für die Programmstruktur gibt es DO und LOOP (entspricht etwa dem FOR. . .NEXT in BASIC), das (;) als Abschluß einer Colon-Definition und andere Strukturen wie IF. . .THEN und BEGIN. . .UNTIL, die für Verzweigungen und Schleifen eingesetzt werden. Auch hier ist es möglich, eigene Programmstrukturwörter zu definieren.

Wie behandelt FORTH nun Eingaben? Symbolgruppen ohne Leerzeichen werden als Wörter interpretiert und nach den folgenden drei Regeln bearbeitet:

1) Wenn FORTH das Wort im Vokabular findet, führt es die entsprechende Definition aus. Kommt das Wort mehrfach vor, wird die zuletzt eingegebene Definition eingesetzt.

2) Steht das Wort nicht im Vokabular, dann prüft FORTH, ob das Symbol (oder die Symbolgruppe) eine Zahl ist. Ist dies der Fall, legt es die Zahlen auf dem „Stapel" ab.

Die Leerzeichen

Da Wörter aus Zeichenfolgen bestehen, sieht FORTH die folgenden Angaben als Wörter an (die jedoch nicht im Vokabular definiert sein müssen):

```
KARTOFFEL
ersehnterLottogewinn
239
pH
25DM
+
"achtung"
,!XXX)hallo-dortQWERTZ
```

Außer beim Leerzeichen versucht FORTH nicht, den Zeichen eine bestimmte Bedeutung zuzuordnen. Sie werden vermutlich Wörter wie LAENGE und PUNKTE für Ihre Variablen einsetzen, doch FORTH würde auch 23PI/2 oder 1BIS10 akzeptieren.

In vielen Sprachen müssen Variablennamen mit einem Buchstaben anfangen und dürfen danach nur aus Buchstaben und Zahlen bestehen. Ursache dafür ist die Interpretation des Leerzeichens. Die meisten Sprachen sehen Leerzeichen lediglich als „Dekoration" an, durch die Programme übersichtlicher werden. Da sie problemlos weggelassen werden können, enthalten mathematische Ausdrücke wie 2+3 in BASIC selten zusätzliche Leerzeichen.

Die Sprachen müssen daher andere Möglichkeiten haben, zwischen den verschiedenen Komponenten eines Programms unterscheiden zu können. Wenn Sie in BASIC eine Variable 2+3 nennen würden, weiß der Interpreter nicht, ob Sie die Variable oder einen mathematischen Ausdruck meinen. Um diese Mehrdeutigkeit auszuschließen, werden Variablennamen Einschränkungen unterworfen.

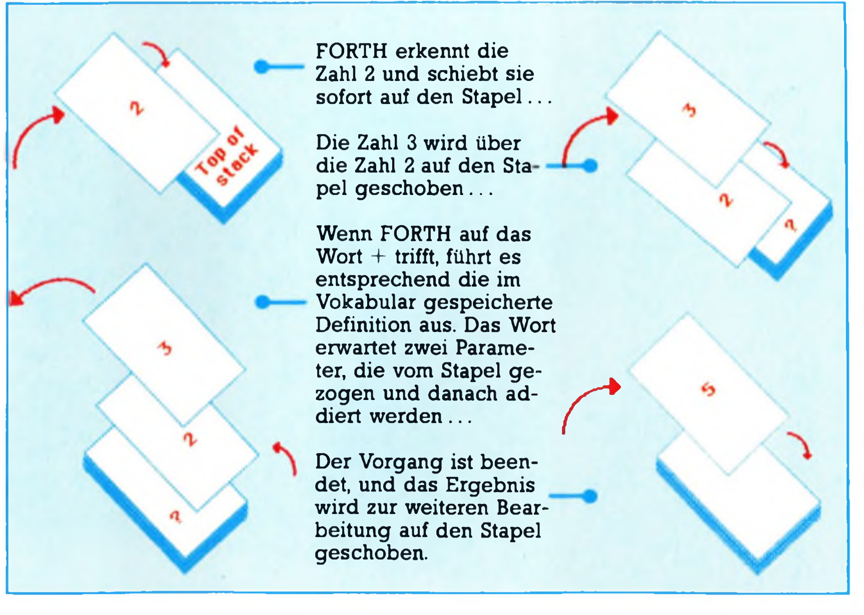
FORTH löst dieses Problem auf andere Weise. Da das Leerzeichen kompromißlos als Trennelement zwischen Wörtern eingesetzt wird, sind keine weiteren Regeln nötig.

FORTH-Dialekte
In figFORTH muß der Wert einer Variablen jeweils vor der Variablen stehen.

```
10 VARIABLE
BLUMENVASEN
```

In diesem Beispiel ist der Wert Zehn.

Wenn FORTH auf ein Wort trifft, führt es den Vorgang aus, der unter diesem Wort im Vokabular gespeichert ist. Trifft es jedoch auf eine Zahl, dann speichert FORTH diese in einem Bereich, der „Stapel" genannt wird. Dort bleibt die Zahl, bis FORTH sich wieder daran „erinnern" muß – beispielsweise bei der Ausführung eines weiteren Befehls. Unser Bild zeigt, wie FORTH die Eingabe 2 3 + interpretiert.



3) Ist die Symbolgruppe weder eine Zahl, noch ein im Vokabular eingetragenes Wort, gibt FORTH eine Fehlermeldung aus.

Bei Colon-Definitionen sind diese Regeln etwas geändert, da die entsprechenden Vorgänge nicht direkt ausgeführt werden können. Wörter wie (:), VARIABLE und CONSTANT sind daher Ausnahmen. Die Schreibweise

```
VARIABLE PUNKTE
```

würde gemäß Regel 3 einen Fehler erzeugen. VARIABLE behandelt PUNKTE jedoch auf eigene Weise und umgeht so die drei Regeln. Wenn dies nicht der Fall wäre, ließen sich keine Wörter in das Vokabular eintragen.

Durch das Vokabular erhält FORTH eine breit angelegte, interaktive Flexibilität. Die drei Regeln stellen sicher, daß neue Definitionen mit den alten Definitionen gleichgestellt sind.

Zurück zum Ursprung

Mit ein wenig Erfahrung in der Maschinencodeprogrammierung ist FORTH leicht zu verstehen. Für Anfänger ohne diese Kenntnisse können die ersten Programmierversuche oft sehr frustrierend sein.

Bei der Definition des Wortes QUADRAT, das als Subroutine auf dem Bildschirm ein Quadrat mit der Seitenlänge SEITE zeichnet, gehen wir folgendermaßen vor. Zunächst definieren wir die Variable SEITE:

```
VARIABLE SEITE
```

FORTH wird dabei angewiesen, Platz zu reservieren, an dem der numerische Wert gespeichert werden kann. Mit @ (holen) können wir von dieser Adresse Werte abrufen und mit ! (speichern) dort ablegen.

```
20 SEITE !
```

ordnet der Variablen SEITE den Wert 20 zu.

Wir können das neue Wort QUADRAT nun mit den „Wörtern" : und ; (ähnlich wie TO und END in LOGO) definieren:

```
:QUADRAT
  SEITE !
    4 0 DO
      SEITE @ FORWARD
    90 RIGHT
  LOOP
```

Diese Prozedur setzt voraus, daß die Wörter RIGHT und FORWARD bereits definiert wurden, bei denen beispielsweise die Funktion

```
50 FORWARD
```

die Turtle 50 Bildschirmpunkte vorwärts bewegt.

Weiterhin muß der Anwender den Wert für SEITE eingeben, so daß

```
50 QUADRAT
```

ein Quadrat mit der Seitenlänge von 50 Bildschirmpunkten zeichnet.

Sehen wir uns einmal an, wie die Subroutine arbeitet, wenn wir 50 QUADRAT eingeben. FORTH untersucht zunächst die Eingabe und stellt fest, daß die erste Symbolgruppe (50) durch ein Leerzeichen von der nächsten (QUADRAT) getrennt ist. FORTH überprüft nun, ob diese Gruppe im Vokabular eingetragen ist. Wenn Sie die Zeichengruppe 50 zuvor nicht als Wort definiert haben, prüft FORTH, ob es sich um eine Zahl handelt. Da dies der Fall ist, legt FORTH die Zahl im Speicher ab und untersucht die nächste Symbolgruppe – QUADRAT

QUADRAT ist natürlich im Vokabular vorhanden (wir haben sie gerade definiert). FORTH ruft nun die Definition auf. Dabei trifft es als erstes auf den Ausdruck SEITE !. Da SEITE ! an dieser Stelle kein Wert zugewiesen wurde, prüft FORTH seinen Speicher und findet dort den von uns eingegebenen Wert (50), den es der Variablen SEITE nun zuordnet.

Im Gegensatz zu Logo überprüft FORTH an dieser Stelle nicht, ob dies der richtige Wert ist, sondern nimmt an, daß wir die Zahl an der korrekten Stelle eingegeben haben. Wenn wir statt 50 QUADRAT nur QUADRAT angeben, erhalten wir keine Fehlermeldung.

4 0 DO. . .LOOP erklärt sich eigentlich von selbst. Interessant ist hierbei, daß die Variable SEITE im Inneren der Schleife durch die Wörter SEITE @ FORWARD an die Subroutine FORWARD übergeben wird. Die Subroutine ruft dabei zunächst die Adresse von SEITE auf und holt sich dann (mit @) den Wert dieser Adresse. 90 RIGHT ist notwendig, um die Turtle nach Abschluß einer Seite um 90 Grad zu drehen und das Quadrat zu vervollständigen.

Die Definition des Wortes QUADRAT zeigt nur einen der vielen Aspekte, die die Flexibilität von FORTH bietet. Es ist nicht nur möglich, „Vokabulare" anzulegen, die mechanische Geräte wie das im Bild gezeigte Teleskop steuern, es lassen sich auch Systeme für abstrakte Anwendungen damit erstellen. So setzt beispielsweise das britische Softwarehaus *Mastertronics* FORTH für die Entwicklung von Abenteuerspielen ein.



Rechnen mit UPN: 2+3 gleich 23+

Gesteuerter Stapel

Es ist nicht immer möglich, den Stapel so aufzubauen, daß alle Zahlen in der richtigen Reihenfolge stehen. FORTH bietet daher eine Reihe von Wörtern, die die Werte des Stapels umstellen. Hier ein paar Beispiele:

- DROP x-- (löscht den obersten Stapelwert)
- DUP x--x, x (dupliziert den obersten Stapelwert)
- SWAP x, y--y, x (tauscht die beiden obersten Stapelelemente gegeneinander aus)
- ROT x, y, z -- y, z, x (rotiert die ersten drei Stapelelemente, so daß die zuunterst liegende Zahl oben liegt)
- OVER x, y -- x, y, x (kopiert die zweitunterste Nummer und legt sie oben auf den Stapel)
- PICK xn, ... x2, x1, x0, n--xn, ..., x2, x1, x0, xn (wie OVER. Sie müssen jedoch eine zusätzliche Zahl (n) angeben, die besagt, welche Zahl kopiert werden soll. 1 PICK entspricht beispielsweise OVER, 2 PICK kopiert die drittunterste Zahl, während 0 PICK wie DUP arbeitet)
- ROLL xn, ..., x2, x1, x0, n -- ..., x2, x1, x0, xn (wie ROT, aber mit einer zusätzlichen Zahl wie PICK. 2 ROLL entspricht ROT, während 3 ROLL vier Zahlen rotiert. 1 ROLL entspricht SWAP).

In FigFORTH fehlen PICK und ROLL. In FORTH-79 sind sie enthalten, doch spricht die Zusatzzahl einen um Eins höheren Wert an, als in FORTH-83. In FORTH-79 entspricht 3 ROLL daher ROT.

In dieser Folge untersuchen wir die Logik der „Umgekehrten Polnischen Notation" (UPN) und wollen zeigen, wie diese Schreibweise mathematische Abläufe und die Bearbeitung von Daten vereinfacht.

Es ist selbstverständlich, daß wir die Summe zweier Zahlen nur erhalten können, wenn wir beide Zahlen kennen. Bei der Eingabe von 2+3 (das "+" steht in der Mitte), wartet ein Computer daher erst auf die zweite Zahl, bevor er sich mit dem + beschäftigt. Die Maschine versteht die Aufgabe als 2 3 +.

Wir sind daran gewöhnt, mathematische Operatoren wie +, –, * und / in der Mitte („Infix Notation") zu schreiben (zum Beispiel 2+2). Die Schreibweise entspricht der Umgangssprache (zwei plus zwei ergibt vier) und trennt die beiden Operanden deutlich voneinander.

Die Infix Notation bringt bei Spracherweiterungen jedoch große Nachteile. Zunächst muß der Operator exakt zwei Operanden (Argumente oder Parameter) haben – auf jeder Seite einen. Werden mehrere Operanden gebraucht, muß eine funktionelle Schreibweise wie FNf(a, b, c, d, e) verwandt werden. Weiterhin sind Ausdrücke wie 2+3*5 nicht eindeutig: Wird hier das + zuerst ausgeführt, oder das *? Das Problem läßt sich nur durch zusätzliche Regeln lösen, die beispielsweise besagen, daß * eine höhere Priorität hat als +. Wenn Sie bei der Infix Notation neue Operatoren einführen wollen, müssen Sie daher auch die Regeln erweitern.

FORTH bietet hier eine Lösung, die den Fähigkeiten des Computers am besten entspricht: Bei allen Operatoren oder Funktionen (also allen Wörtern) werden zuerst die Operanden geschrieben und dann das Wort (beispielsweise 2 3 +). Die Schreibweise funktioniert wie ein Kochrezept: Zuerst holen Sie alle Zutaten, und dann verarbeiten Sie diese. Die technische Bezeichnung dieser Methode lautet „Umgekehrte Polnische Notation" (UPN). In der Direkten Polnischen Notation steht der Operator vor den Operanden. LOGO verwendet diese Schreibweise bei neuen Funktionen.

Nehmen wir an, Sie haben einen Ausdruck mit Infix Notation wie 2*3+8/4. Da nach den üblichen Prioritätsregeln das + zuletzt ausgeführt wird, ist das Endergebnis die Summe von 2*3 und 8/4. Für die UPN-Schreibweise wird

der Ausdruck umgeformt:
(2*3) (8/4) +

Hier haben jedoch 2*3 und 8/4 noch die Infix Notation. Die zweite Umformung ergibt daher:
(2 3 *) (8 4 /) +

Die UPN braucht keine Klammern, da sie in jedem Fall eindeutig ist. FORTH setzt Klammern außerdem nur für Kommentare ein. Hier das endgültige UPN-Format von 2*3+8/4:
2 3 * 8 4 / +

Bedenken Sie, daß in FORTH die Leerzeichen unbedingt notwendig sind.

Die Umgekehrte Polnische Notation kann also die beiden Probleme der Infix Notation lösen: Bei der UPN sind pro Operator nicht nur zwei Operanden möglich, sondern mehrere (gefolgt von einem Operator). Beispielsweise hat in FORTH der Operator */ drei Operanden: Er multipliziert die ersten beiden und dividiert das Ergebnis durch den dritten Operanden.

Das zweite Problem der Infix Notation sind die Prioritätsregeln und Klammern, durch die die Reihenfolge der Einzelberechnungen festgelegt wird. Die UPN kommt ohne Regeln und Klammern aus – sie teilt dem Computer exakt mit, wie das Ergebnis berechnet wird.

Ergebnis im Speicher

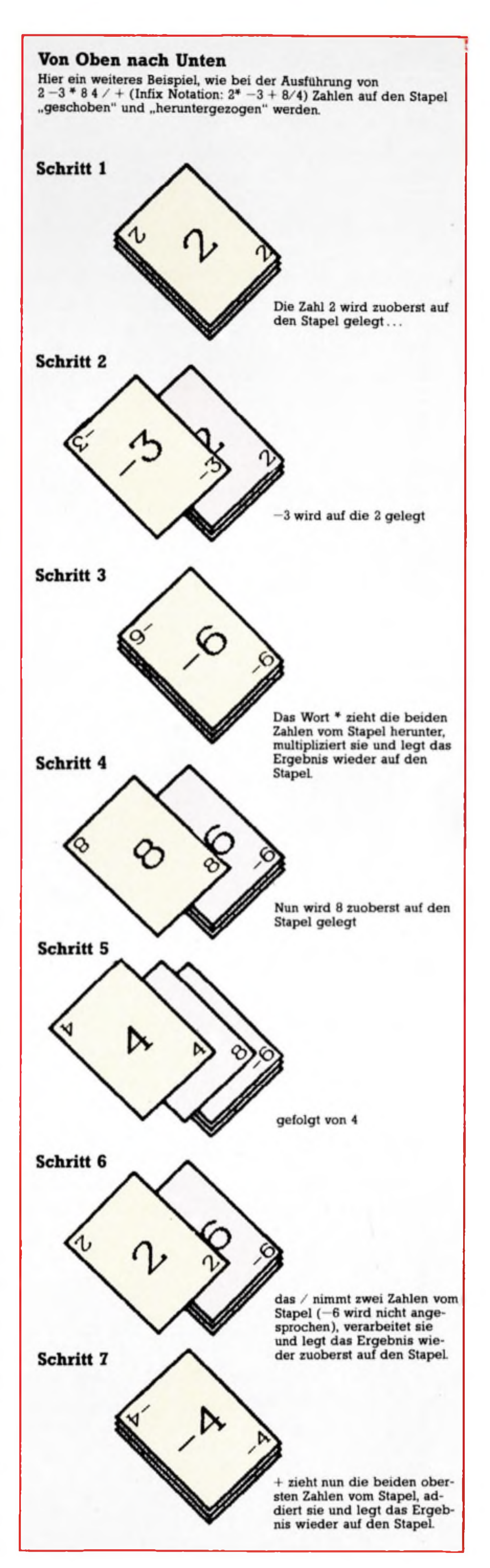
Sehen wir uns an dem einfachen Beispiel 2 3 + einmal an, wie die UPN ausgeführt wird. FORTH trifft zuerst auf die 2 und speichert sie. Danach findet es die 3, die ebenfalls gespeichert wird. Bei + nimmt der Computer an, daß er sich bereits zwei Zahlen „gemerkt" hat. Er findet sie, addiert sie und legt das Ergebnis wieder im Speicher ab. Dort steht es für die weitere Bearbeitung zur Verfügung.

In der letzten Folge erwähnten wir kurz, daß FORTH sich an Zahlen „erinnert", indem es diese auf den Stapel schiebt (nach der LIFO-Methode – Last In First Out). Dieses Konzept ist Maschinencodeprogrammierern sicherlich vertraut, doch kann es für Anfänger schon recht ungewohnt sein. Wir geben daher hier ein weiteres Beispiel. Stellen Sie sich einen Stapel Spielkarten vor. Um eine Nummer zu „speichern", schreiben Sie sie auf eine neue Karte, die Sie oben auf den Stapel legen. Für das + nehmen Sie die beiden obersten Karten herunter und addieren die beiden darauf notierten Zahlen, verändern jedoch nicht den Rest des Stapels. Der Kasten zeigt die Entwicklung des Stapels bei der Bearbeitung von 2 −3 * 8 4 /+ ((2 * −3) + 8 / 4)). Beachten Sie, daß in Schritt 6 die Karte −6 von der Teilung 8 / 4 nicht berührt wird.

Zwar zeigt dieser Ablauf, wie der Stapel mit Ganzzahlen funktioniert, in FORTH werden Variablen jedoch als Adressen und nicht als Ganzzahlen behandelt. Die Adressen werden

Von Oben nach Unten

Hier ein weiteres Beispiel, wie bei der Ausführung von 2 −3 * 8 4 / + (Infix Notation: 2* −3 + 8/4) Zahlen auf den Stapel „geschoben" und „heruntergezogen" werden.



Die Zahl 2 wird zuoberst auf den Stapel gelegt ...

−3 wird auf die 2 gelegt

Das Wort * zieht die beiden Zahlen vom Stapel herunter, multipliziert sie und legt das Ergebnis wieder auf den Stapel.

Nun wird 8 zuoberst auf den Stapel gelegt

gefolgt von 4

das / nimmt zwei Zahlen vom Stapel (−6 wird nicht angesprochen), verarbeitet sie und legt das Ergebnis wieder zuoberst auf den Stapel.

+ zieht nun die beiden obersten Zahlen vom Stapel, addiert sie und legt das Ergebnis wieder auf den Stapel.

eingesetzt, um (mit @, dem Befehl für „holen") die Variablenwete anzugeben oder (mit !, dem Befehl für „speichern"), um geänderte Werte auf die Adressen zu schreiben.

Unser Beispiel zeigt deutlich, wie exakt die Umgekehrte Polnische Notation arbeitet. FORTH unterhält eine eigene Version eines „Kartenstapels", so daß der Computer bei jedem Wort die entsprechenden Abläufe ausführen kann, ohne sich darum kümmern zu müssen, was zuvor oder danach passiert.

Hier ein weiteres Beispiel für den Einsatz des Stapels: Das Wort 2* ersetzt den letzten Wert des Stapels durch den doppelten Wert dieser Zahl. Es ist folgendermaßen definiert:

```
:2* (n —— n*2)
2 *
;
```

Beachten Sie, daß Klammern nur Kommentare umschließen. (Das Symbol —— wird in dem nebenstehenden Kasten erklärt.) Hier ein Beispiel für das neue Wort 2*:

```
19 2*.
```

zeigt als Ergebnis 38. Das Symbol . ist das FORTH-Wort für PRINT. Es nimmt den obersten Wert des Stapels und zeigt ihn auf dem Bildschirm an. Das *-Zeichen in der Definition von 2* erwartet mindestens zwei Zahlen. Die zweite Zahl ist dabei die in der Definition enthaltene 2, während sich die erste (in unserem Beispiel 19) bereits auf dem Stapel befinden muß, wenn 2* eingesetzt wird.

Stapelvorgänge dieser Art haben den Vorteil, daß sie mehr als ein Ergebnis erzeugen können. So beläßt das Wort /MOD beispielsweise zwei Werte auf dem Stapel: den „Quotienten" (das Ergebnis) und den „Rest" einer Division. Mit —— läßt sich dieser Vorgang aber auch folgendermaßen ausdrücken:

```
m, n —— Rest, Quotient von m/n
```

Bei der Infix Notation wäre dieser Vorgang nicht möglich, bei der UPN ist er normal.

Die direkte Programmierung ist zwar eine der beeindruckendsten Eigenschaften von FORTH, kann aber auch Probleme verursachen. So stehen die Zahlen des Stapels zuweilen nicht in der richtigen Reihenfolge und müssen umorganisiert werden. Der nebenstehende Kasten enthält einige Standardwörter, die für diesen Zweck eingesetzt werden. Das nachfolgend definierte Wort */ ist zwar nicht so gut wie die Standarddefinition (die auch die richtige Antwort bietet, selbst wenn das Ergebnis zu groß für den Stapel ist), zeigt aber, wie ROT und SWAP funktionieren:

```
: */ (x,y,z——x*y/z)
  ROT ROT *
  SWAP /
;
```

Hier der Ablauf am Beispiel von 4 3 6 */:

Vorgang */	Stapel (oben → unten)
Am Anfang von */	4,3,6
ROT	3,6,4
ROT	6,4,3
*	6,12
SWAP	12,6
/	2
	Leer (2 wird dargestellt)

Normalerweise werden die Befehle zur Stapelmanipulation jedoch nur im Inneren von Wortdefinitionen eingesetzt.

Vorsicht

Sie müssen sorgfältig darauf achten, genau die Anzahl an Werten auf den Stapel zu schieben, die das Wort benötigt. Wenn Sie zu wenig anlegen, setzt das Programm selbständig Werte ein. Wurden zu viele eingesetzt, dann stimmt das Ergebnis ebenfalls nicht.

Für die genaue Wort-Definition können Sie die Schreibweise "——" einsetzen. Dabei wird die Liste der Werte, die ein Wort braucht, durch "——" von den Werten getrennt, die es auf dem Stapel zurückläßt. Hier einige Beispiele:

Wort:	Auswirkung:
+	m,n——m+n
.	m ——
*/	x, y, z——x*y/z

Die Schreibweise "——" ist kein Bestandteil von FORTH, sondern soll nur die Schreibweise übersichtlicher machen. Bei der Verwendung von Klammern sollten Sie beachten, daß nach der ersten Klammer ein Leerzeichen stehen muß. Die Klammer ist ein Wort, das besagt: Ignoriere von hier an alles, bis eine geschlossene Klammer auftaucht.

Abfrage des Stapels

Ein sehr praktisches Wort ist .S, das den Stapelinhalt anzeigt. Es ist im Standard-FORTH zwar nicht enthalten, in FORTH-83 aber definiert als:

```
: .S (——)
    (zeigt den Stapel von unten nach oben
      an)
    0 DEPTH 1 — DO
      I PICK.
    —1 + LOOP
;
```

In FORTH-79 funktioniert PICK anders. Die dritte Zeile muß dann lauten

```
1 DEPTH DO
```

DEPTH (—— Stapeltiefe) informiert Sie darüber, wie viele Zahlen auf dem Stapel abgelegt waren, bevor Sie DEPTH aufgerufen haben. Da FigFORTH jedoch weder über DEPTH noch über PICK verfügt, funktionieren diese Definitionen dort natürlich nicht.

Zwei Bedingungen

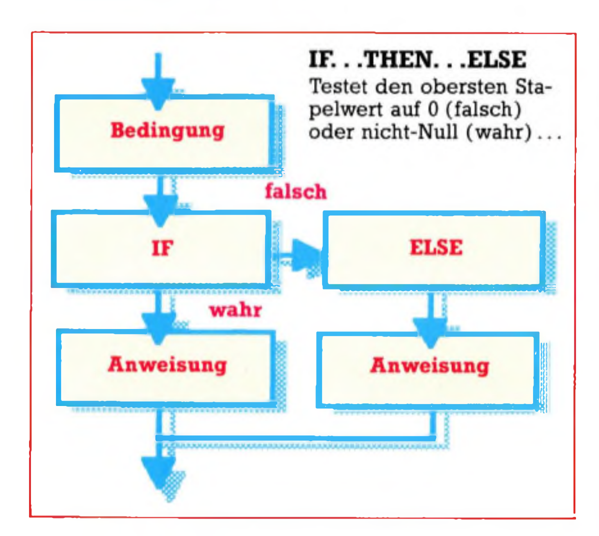
In unserer FORTH-Serie haben wir bisher einfache, gradlinige Routinen behandelt, aber keine Strukturen, mit denen sich der Programmfluß steuern ließe. In dieser Folge sehen wir uns mehrere dieser Strukturen an, darunter IF. . .THEN. . .ELSE und die DO-Schleife.

Wie jede andere Programmiersprache besitzt auch FORTH eine ganze Reihe Strukturen, die den Programmfluß steuern. Wenn Sie Sprachen wie PASCAL, C oder die hochentwickelten BASICs bereits kennen, werden Sie die folgenden Abläufe leicht verstehen. Durch die Mechanik des FORTH-Stapels sehen die Strukturen jedoch vielfach „seitenverkehrt" aus.

Steuerstrukturen können in FORTH nur innerhalb von Colon-Definitionen eingesetzt werden, da die Verzweigungsbefehle erst in bedingte und absolute Sprünge compiliert werden müssen. Dieser Vorgang bräuchte aber viel Zeit, die dann während der Programmausführung nicht zur Verfügung stehen würde. Deshalb optimiert FORTH die Abläufe schon beim Eintrag ins Vokabular.

Hier die Abläufe des Standard-FORTH:

- Bedingung IF Anweisung wahr ELSE Anweisung falsch THEN



Wenn die Bedingung „wahr" ist, wird „Anweisung wahr" ausgeführt. Ist sie „falsch", die „Anweisung falsch". Wie in den meisten Computersprachen kann der ELSE-Teil und die „Anweisung falsch" auch weggelassen werden. In diesem Fall geht FORTH sofort auf die Anweisung hinter THEN über, wenn die Bedingung „falsch" ergibt.

Die Bedingung und die beiden Anweisungen (wahr und falsch) können beliebig viele Forth-Wörter sein, sogar andere IF. . .THEN. . . ELSE sind möglich. Die Bedingung wird dabei auf den Stapel geschoben und von IF wieder heruntergezogen. Hier ein Beispiel, wie angezeigt wird, ob eine Zahl des Stapels gerade oder ungerade ist:

```
:PARITAET    ( n--)
             ( zeigt an, ob n "gerade" oder
               "ungerade" ist)
  DUP.       ( zeigt n an)
  ."ist"
  2 MOD 0=IF
    ."gerade"
  ELSE
    ."ungerade"
  THEN
;
```

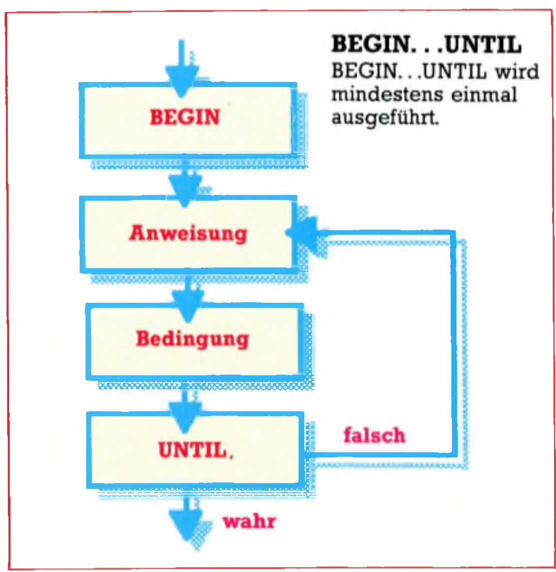
MOD liefert bei der Teilung der ersten Zahl durch die zweite den Rest.

- BEGIN. . .UNTIL

führt eine Schleife aus, solange die Bedingung wahr ist. Die meisten modernen Computer und auch die neueren BASIC-Dialekte verfügen über eine ähnliche Struktur.

BEGIN Anweisung Bedingung UNTIL

Es werden die Schleife und die Bedingung ausgeführt. Die Bedingung ist das Ende des Schleifenteils und beläßt einen Wert auf dem Stapel, mit dem UNTIL weiterarbeiten kann. Der Schleifenteil muß daher mindestens einmal ausgeführt werden:

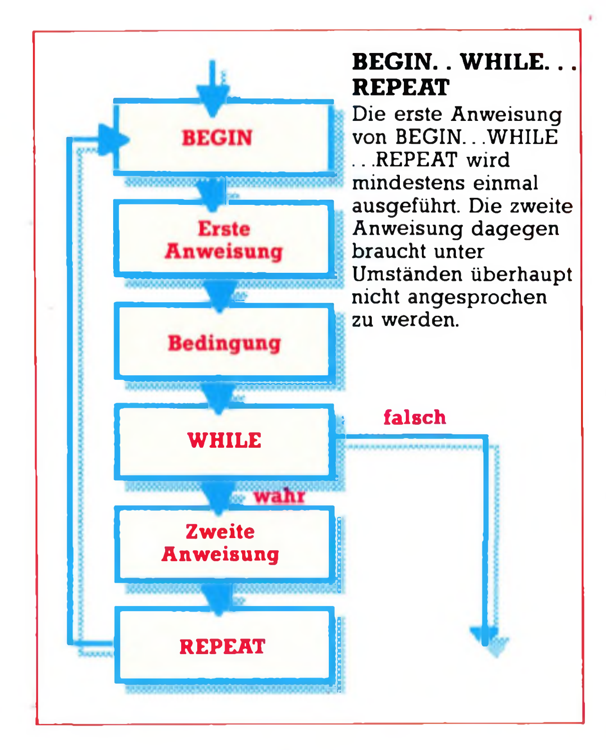


```
: HOCH2       ( --)
              ( zeigt alle Quadratzahlen von
                2 an, die unter 10000 liegen)
  1           ( Erste Hochzahl für 2)
  BEGIN
    CR DUP.( Quadratzahl auf eine neue
      Zeile setzen)
    2* (nächste Quadratzahl von 2 holen)
  DUP 1000 > UNTIL (testen, ob die Zahl
    schon zu groß ist)
  DROP ( letzte Quadratzahl nicht anzeigen)
```

● BEGIN. . .WHILE. . .REPEAT

WHILE ist flexibler als UNTIL, da Sie im Inneren des Schleifenteils entscheiden können, ob die Schleife beendet werden soll, und Sie nicht erst das Ende abwarten müssen.

BEGIN 1. Schleife Bedingung WHILE 2. Schleife REPEAT

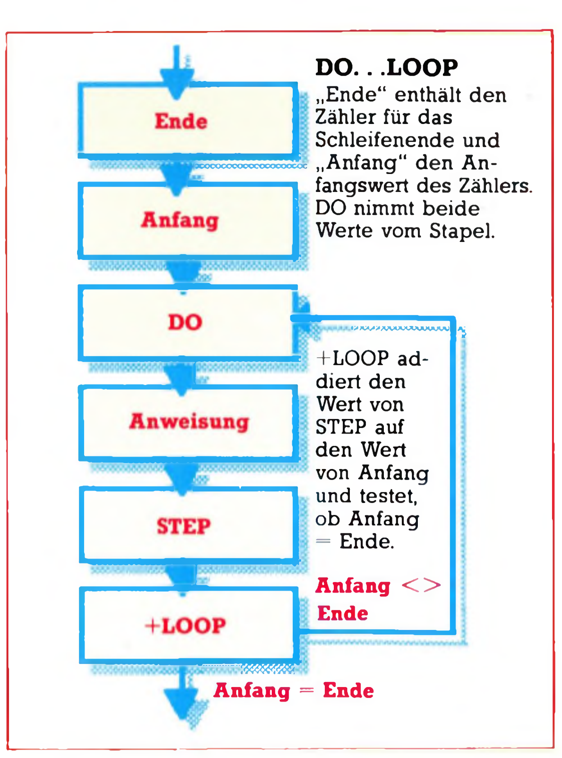


Wie in UNTIL wird auch hier die 1. Schleife zumindest einmal ausgeführt und die Bedingung abgefragt. An dieser Stelle gibt es jedoch zwei wesentliche Unterschiede zu UNTIL. Wenn die Bedingung „falsch" ergibt, wird die 2. Schleife übersprungen und damit die Schleife beendet. Ergibt sie „wahr", führt FORTH die 2. Schleife aus, bevor sie wieder auf BEGIN springt.

● DO. . .LOOP

BASIC	**FORTH**
FOR X=Anfang TO Ende	(Ende+1) Anfang DO
Schleifeninhalt	Schleifeninhalt
NEXT X	LOOP
FOR X=Anfang TO Ende STEP Step	(Ende+1) Anfang DO
Schleifeninhalt	Schleifeninhalt
NEXT X	Step +LOOP

Die DO-Schleife in FORTH entspricht den FOR-Schleifen in BASIC ubd vielen anderen Sprachen. Die oben angeführte tabellarische Gegenüberstellung zeigt die auffälligen Unterschiede zwischen FORTH und BASIC.



Zunächst steht der Anfangs- und Endwert vor dem DO, damit das DO sie vom Stapel nehmen kann. Ende wird deshalb auch zuerst aufgeführt. Es ist allerdings weniger deutlich, warum Ende um eins größer sein muß als die Anzahl der Schleifendurchläufe. Die Ursache liegt darin, daß 4 0 DO die Schleife mit den Werten 0, 1, 2 und 3 (also insgesamt viermal) durchläuft, nicht aber mit dem Wert 4.

Wie in BASIC ist STEP möglich. Wird die Angabe weggelassen, nimmt LOOP als STEP automatisch 1 an. Wird STEP jedoch angegeben, muß +LOOP statt LOOP eingesetzt werden,

Bedingungen

Die folgenden Wörter eignen sich für den Einsatz mit IF, UNTIL und WHILE. Ihre Ergebnisse werden auf dem Stapel abgelegt und sind in FORTH-83 entweder -1 (wahr) oder 0 (falsch). Ältere FORTH-Versionen verwenden 1 für wahr und 0 für falsch.

IF, UNTIL und WHILE benötigen nicht die Angaben 0, 1 oder -1. 0 zeigt einfach falsch an, während jede andere Zahl wahr ist.

=	m,n--wahr oder falsch (wahr IF m = n)
<	m,n--wahr oder falsch (wahr IF m < n)
>	m,n--wahr oder falsch (wahr IF m > n)
0=	m,n--wahr oder falsch (wahr IF m = 0)
0<	m,n--wahr oder falsch (wahr IF m < 0)
0>	m,n--wahr oder falsch (wahr IF m > 0)
NOT	wahr oder falsch--wahr oder falsch
AND	m,n--m AND n
OR	m,n--m OR n

wobei STEP unmittelbar vor +LOOP steht.

In FORTH gibt es keine Steuervariable (wie das X in BASIC). Statt dessen schiebt das Wort I den Schleifenwert auf den Stapel. Wenn mehrere DO-Schleifen ineinander verschachtelt sind, bezieht sich I immer auf die innerste und J auf die nächstäußere. Auf diese Weise ist sichergestellt, daß bei mehrfach verschachtelten Schleifen die entsprechenden Werte intakt sind und korrekt gezählt werden.

Hier ein Programmbeispiel, das eine Zahl mit einer anderen potenziert. Da FORTH im Gegensatz zu BASIC keinen Standardoperator dieser Art besitzt, müssen Sie ihn selbst definieren:

```
: * * ( m,n--m**n)
  DUP 0 < IF ( IF n < 0 ist das Ergebnis 0)
    DROP DROP 0
  ELSE
    DUP 0 = IF ( IF n=0 ist das Ergebnis 1)
      DROP DROP 1
    ELSE
      1 ( multipliziert n mal mit m)
      SWAP 0 DO ( Schleife n mal durch-
                  laufen)
        OVER * ( ersten Wert des Stapels
          mit m multiplizieren)
        LOOP
        SWAP DROP ( m löschen)
      THEN
    THEN
```

Hier sind die Sonderfälle etwas kompliziert. Ist die Hochzahl negativ, dann ist die korrekte Antwort 1 dividiert durch das potentierte Ergebnis. Da FORTH nur mit Ganzzahlen arbeitet und sich mit diesem Programm die Teilung nicht exakt ausführen läßt, ist das Ergebnis jedoch 0. Der Fall n=0 ist subtiler. Wenn 1 nullmal mit m multipliziert wird (die Schleife also nullmal durchläuft), sollte das Ergebnis 1 lauten. Da FORTH seine Schleifen jedoch mindestens einmal durchläuft, muß dieser Fall besonders berücksichtigt werden. In der Praxis gibt es viele Situationen, in denen die Ausführung einer DO-Schleife nicht gewünscht wird und die abgefangen werden müssen.

Hier ein anschauliches Beispiel mit einer verbesserten Version von HOCH2. Dabei wird außer dem Schleifenwert I auch das Wort LEAVE eingesetzt, das die Schleife unterbricht und sofort auf die Anweisung hinter LOOP oder +LOOP springt:

```
: HOCH2      ( n--)
             ( zeigt Exponenten und Potenzen
             von 2 an, solange die Potenzen
             kleiner als n sind)
  15 0 DO ( 2 ** 14 ist die größte Zahl, die sich
             anzeigen läßt)
    DUP 2 I * * < IF ( IF 2**I > n)
      LEAVE
    ELSE
      CR I. 2 I * *
    THEN
  LOOP
  DROP ( löscht n)
```

Bei diesen Programmstrukturen kommen Sie ohne Zeilennummern und GOTO aus. Wenn Sie an BASIC gewöhnt sind, kann der Ablauf anfangs etwas ungewohnt sein. Zeilennummern sind jedoch nur eine Eigenschaft von BASIC und haben nur wenig mit der allgemeinen Programmplanung zu tun.

Hätten Sie das DROP am Programmende berücksichtigt, wenn Sie HOCH2 selbst geschrieben hätten? Hier zeigt sich, daß der Einsatz des Stapels genau beobachtet werden muß.

Diese Aufgabe können Sie sich jedoch erleichtern, wenn Sie Ihre Wortdefinition so kurz wie möglich halten und die Abläufe unterteilen – wie in der zweiten Definition von HOCH2. Obwohl die neue Definition flexibler ist – Sie können einen Endwert eingeben und sind nicht an 10000 gebunden – ist sie nicht komplizierter. Wenn Sie dagegen die erste Version verändert hätten, wären Sie schon nach kurzer Zeit mit allen Arten von SWAP und OVER konfrontiert worden. Da Sie ** definiert und damit zwei kurze Wörter haben, ist die zweite Version wesentlich einfacher.

Druckfunktion

Strings im Inneren einer Colon-Definition lassen sich mit folgendem Format ausdrucken: . "String"
Auf ." muß mindestens ein Leerzeichen folgen, da es ein Wort ist. Weitere Leerzeichen werden als Teil des String angesehen.
Der String steht hinter dem ." (und folgt damit nicht der Umgekehrten Polnischen Notation), da der Stapel keine Strings verarbeiten kann. Für die Verarbeitung von Stringvariablen gibt es im Standard-FORTH keine Abläufe. Sie können die Sprache jedoch leicht um diese Möglichkeit erweitern.
Das Wort CR (Return) spricht auf dem Bildschirm eine neue Zeile an.

Grenzen von DO

Obwohl das Grundkonzept von DO recht einfach ist, gibt es einige Merkwürdigkeiten: Wenn STEP (vor +LOOP) negativ ist, kann der Wert von LOOP bereits das Ende erreichen, ohne davor anzuhalten. 2 0 DO...-1 +LOOP durchläuft daher die Werte 0, -1 und -2.

Wir haben weiterhin gesehen, daß DO-Schleifen mindestens einmal ausgeführt werden, selbst 0 0 DO...LOOP. FORTH-83 hat hier eine Überraschung parat: Es führt diese Schleife 65.536 mal aus, -1 0 DO...LOOP 65.535mal und -2 0 DO...LOOP 65.534mal etc.

Hier wäre es am leichtesten, überhaupt nicht mit diesen Werten zu arbeiten. Wenn Sie jedoch wissen, wie der Computer Ganzzahlen mit und ohne Vorzeichen verarbeitet, wird Ihnen auch der Grund dafür deutlich sein. Sobald der Anfangswert hinter Ende liegt, läuft der Zähler bis 32.768. Diese Zahl wird dann als -32.768 angesehen und aufwärtsgezählt, bis Ende von unten her erreicht ist.

Wenn bei älteren FORTH-Versionen der Anfangswert das Ende überschritten hat, wird die Schleife nur einmal ausgeführt. Diese Versionen behandeln jedoch auch LEAVE geringfügig anders. Hier springt das Programm nicht aus der Schleife heraus, sondern stellt sicher, daß auch beim nächsten LOOP oder +LOOP ein Ausstieg möglich ist.

Gesiebter Zahlensalat

FORTH kann nur Ganzzahlen und Festkommaformate verarbeiten. Wir untersuchen diese Eigenart am Beispiel des „Sieb des Eratosthenes", das wir schon einmal in der PASCAL-Serie abgedruckt hatten.

Bruchzahlen behandelt FORTH im „Festkommaformat". Die eingesetzten Zahlen werden durch Konstante – normalerweise Zehnerpotenzen – dividiert oder multipliziert.

Die Zahlen des FORTH-Stapels haben je 16 Bits und können entweder vorzeichenbehaftete Ganzzahlen (das Vorzeichen wird vom höchstwertigen Bit angezeigt) zwischen −32768 und +32767 darstellen oder Ganzzahlen ohne Vorzeichen zwischen 0 und 65535. Am einfachsten läßt sich dies durch die Eingabe

```
65535.
```

feststellen. Die Antwort ist −1, da die oben auf dem Stapel liegende Zahl als vorzeichenbehaftet angesehen wird. Die Zahl 65535 ohne Vorzeichen wird auf dem Stapel von dem gleichen Wert dargestellt wie −1 mit Vorzeichen.

Das Wort U. funktioniert wie. und sieht den obersten Stapelwert ohne Vorzeichen an.

```
65535 U.
```

zeigt daher 65535.

Etliche Sprachversionen setzen für diesen Zweck andere Wörter ein. Die Wahl zwischen Vorzeichen oder keinem Vorzeichen wird bei Zahlen zwischen 32768 und 65535 wichtig. Sie müssen sich dann entscheiden, ob Sie die Wörter für vorzeichenbehaftete Zahlen einsetzen wollen (in diesem Fall verursachen Zahlen über 32768 einen Überlauf) oder Zahlen ohne Vorzeichen (in diesem Fall entstehen keine negativen Zahlen). Ein gutes Beispiel dafür sind Werte, die Speicheradressen anzeigen und nur mit Wörtern ohne Vorzeichen eingesetzt werden können – zum Beispiel U< statt <. Sie können aber Stapelzahlen auch als Muster im 16-Bit-Format ansehen und damit normale Boolesche Bitoperationen vornehmen.

Unser Programmbeispiel zeigt mit dem Sieb des Eratosthenes alle Primzahlen bis 65536 an. Wie bei allen FORTH-Programmen läßt sich das Programm rückwärts leichter lesen: Fangen Sie mit dem wichtigsten Wort PRIMES an, und suchen Sie die Wortdefinitionen, die mit diesem Wort arbeiten. PRIMES schreibt zwar als erstes etwas auf den Bildschirm, muß aber als letztes – hinter den untergeordneten Wörtern – eingegeben werden.

Das Sieb des Eratosthenes funktioniert folgendermaßen: Sie schreiben alle Zahlen bis zu einer bestimmten Grenze nieder. Wenn Sie eine Primzahl finden, streichen Sie alle Vielfache dieser Zahl durch, da sie keine Primzahlen sein können. Die nächste Primzahl ist dann der nächste Wert, der nicht ausgestrichen wurde. Nach 1 (die aus verschiedenen Gründen nicht als echte Primzahl gilt) finden Sie die erste Primzahl 2 und streichen alle Vielfache davon aus. Die nächste nicht gestrichene Zahl ist nun die Primzahl 3, von der Sie wiederum alle Vielfache streichen etc.

Unser FORTH-Programm speichert acht KByte – oder 65536 Bits – je ein Bit pro Zahl zwischen 1 und 65536. Die Bits stehen zunächst auf 0, werden aber beim Ausstreichen der Zahl auf 1 gesetzt. Der Befehl

```
CREATE BITS 8192 ALLOT
```

reserviert den Speicherplatz für die Bits. (Wir haben 8192 durch die Konstante BYTES ersetzt.) Wir werden später untersuchen, wie dies funktioniert. Nun ist ein Block von 8192 Bytes irgendwo im Speicher reserviert und das neue Wort BITS angelegt, das die Adresse des ersten Bytes auf den Stapel schiebt. Der Stapel enthält jetzt ein Array von Bytes.

Variable CURBYTES

Für das Ansprechen eines dieser Bits brauchen wir zwei Zahlen – eine Zahl gibt an, welches Byte das Bit enthält (wir können für diese Zahl die Speicheradresse des Bytes einsetzen) und die andere, welches Bit innerhalb des Bytes gemeint ist (also ein Wert zwischen 0 und 7). Es ist nicht allzu schwierig, zwischen diesem Zahlenpaar und der Zahl (zwischen 1 und 65536) umzuschalten, die das Bit darstellt.

CURBYTES ist eine Variable. Ihr Wert CURBYTES @ ist die Adresse eines Bytes im Array BITS, CURBYTES @ C @ dagegen der Wert des Bytes. Wenn Sie die Programmiersprache C kennen, wird Ihnen dieses Konzept vertraut sein. Da Primzahlen und Adressen recht groß werden können, setzen wir gelegentlich Wörter ohne Vorzeichen wie U. und U< ein.

Das Wort:

```
+! ( n, Adresse — )
```

tauscht den Wert der angegebenen Adresse gegen die zuvor dort gespeicherte Zahl aus, auf die n addiert wurde. Es eignet sich daher für die Erhöhung eines Variablenwertes um

Zahlenbasis
In FORTH läßt sich die Zahlenbasis austauschen. Die eingegebenen Zahlen können daher binär, oktal, dezimal oder hex sein oder jede andere Basis haben. Das Format wird über BASE angegeben:

```
54 16 BASE ! . DECIMAL
```

zeigt die Hexzahl 36 (54 Dezimal) an und schaltet mit dem Wort DECIMAL auf die Basis 10 zurück. Es ist einfacher, Wörter wie

```
:HEX 16 BASE !
```

zu definieren. Beachten Sie, daß hier eine Zahl Teil einer Wortdefinition ist und von späteren Änderungen der Zahlenbasis nicht beeinflußt wird.

Das Sieb des Eratosthenes, nach dem griechischen Mathematiker benannt, der es entwickelte, ist eine wirkungsvolle Methode, die Vielfachen einer Zahlenfolge „auszusieben", bis nur die Primzahlen im „Sieb" zurückbleiben.

eine bestimmte, zuvor genau definierte Zahl.

Einige Wörter setzen die Arithmetik mit doppelten Zahlenlängen ein. Dabei stellen zwei Zwei-Byte-Zahlen eine Vier-Byte-Zahl dar. Die höherwertige Zahl liegt näher an der Stapelspitze, die niederwertige halb darunter.

Obwohl der Standard von FORTH-83 nur die Eingabe von Zahlen einfacher Länge zuläßt, kann in den meisten Implementationen mit einem Punkt (an jeder Position der Zahl) angezeigt werden, daß die Zahl doppelte Länge besitzt (der Punkt wird dabei natürlich nicht als Dezimalpunkt interpretiert). 10.000 ist beispielsweise die doppelte Zahlenlänge von 10 000 und belegt zwei Stapeleinträge – unten 10 000 und oben 0.

```
8192  CONSTANT  BYTES
CREATE BITS BYTES ALLOT BITS BYTES
+ CONSTANT BITSEND
(  1 bit for each number 1 - 65536 )
VARIABLE  CURBYTE   VARIABLE  CURBIT
(  show number being looked at.  CURBYTE )
(  points byte in BITS, CURBIT to bit  )
VARIABLE  MASK  (  2**CURBIT  )
VARIABLE  PRIME  VARIABLE  PRIME/8
VARIABLE  PRIMEMOD8
(  show prime when found  )
:  SETUP  (  --  )  ( initialises variables )
   BYTES  0  DO  ( zero array BITS )
      0  BITS  I  +  C!
   LOOP
   BITS  CURBYTE  !  0  CURBIT  !  1  MASK  !
;
:  NEXTBIT  (  -- 0 or non-zero  )
(  adjusts CURBYTE etc to find next bit  )
(  and stacks it  )
   1  CURBIT  +!
   MASK  @  2*  MASK  !
   CURBIT  @  8  =  IF
      0  CURBIT  !
      1  MASK  !
      1  CURBYTE  +!
   THEN
   CURBYTE  @  C@  MASK  @  AND
;
:  PRIMEETC ( -- ) ( sets up PRIME, PRIME/8 )
(  and PRIMEMOD8 )
   CURBIT  @  1+
   DUP  8  =  IF
      DROP  0  PRIMEMOD8  !  1
   ELSE
      PRIMEMOD8 !  0
   THEN
   CURBYTE  @  BITS  -  +  PRIME/8  !
   PRIME/8  @  8  *  PRIMEMOD8  @  +  PRIME  !
;
:  ANOTHER-PRIME  (  --  0 or -1  )
(  adjusts CURBYTE etc to next prime, sets )
(  PRIME etc. 0 left if end of BITS reached )
   BEGIN
   NEXTBIT  0=  UNTIL
   CURBYTE  @  BITSEND  U<  IF
      PRIMEETC
      -1
   ELSE
      0
   THEN
;
VARIABLE  DELBYTE  VARIABLE  DELBIT
:  DELMASK  (  2**DELBIT  )
   1
   DELBIT  @  IF   (  if DELBIT non-zero  )
      DELBIT  @  0  DO  (  multiply 1 by 2  )
         2*              (  2 DELBIT times  )
      LOOP
   THEN
;
:  ANOTHER-MULTIPLE  (  --  0 or -1  )
(  adjusts DELBYTE and DELBIT  )
(  to next bit if set )
(  0 left if end of BITS array reached )
   PRIMEMOD8  @  DELBIT  +!
   PRIME/8  @  DELBYTE  +!
   DELBIT  @  7  >  IF
      -8  DELBIT  +!
      1  DELBYTE  +!
   THEN
   DELBYTE @ BITSEND U< BITS
   1- DELBYTE @ U< AND
(  careful in case DELBYTE exceeds 65535 )
;
:  DELETE-MULTIPLES  (  --  )
(  sets bits for m/tiples of current prime )
   CURBYTE  @  DELBYTE  !
   CURBIT  @  DELBIT  !
   BEGIN
   ANOTHER-MULTIPLE  WHILE
    DELBYTE  @  C@  DELMASK  OR  DELBYTE  @  C!
   REPEAT
;
:  PRIMES  (  --  )
(  prints all primes between 2 and 65535 )
   SETUP
   BEGIN
   ANOTHER-PRIME WHILE
      PRIME @ U.
      DELETE-MULTIPLES
   REPEAT
;
```

Mathematische Worte

Addition und Subtraktion: Es gibt keinen Unterschied zwischen Addition und Subtraktion mit oder ohne Vorzeichen. + und – stellen einfache Längen dar, D+ und D– doppelte Längen.
Vorzeichen-Minus: NEGATE für einfache Längen, DNEGATE für doppelte Längen. fig-FORTH nimmt dafür MINUS und DMINUS.
Multiplikation: * hat einfache Länge, mit oder ohne Vorzeichen. UM* hat nicht-vorzeichenbehaftete „gemischte" Länge – der Befehl zieht zwei Zahlen ohne Vorzeichen vom Stapel und legt dort das Ergebnis in doppelter Länge ab. In älteren FORTH-Versionen wird UM* auch U* genannt. Es gibt dort auch das vorzeichenbehaftete Produkt M*.
Division: /, MOD und /MOD haben einfache Länge mit Vorzeichen. Sie ergeben entweder den Quotienten, den Rest oder beide (mit dem Quotienten oben auf dem Stapel). UM/MOD ist die Version von /MOD für gemischte Längen ohne Vorzeichen. Sein Dividend hat dabei doppelte Länge. In älteren FORTH-Versionen ist dies U/ oder U/MOD. FORTH-83 rundet immer auf die nächst kleinere Ganzzahl. 10/3 wird daher auf 3 abgerundet, –10/3 jedoch auf –4. Ältere FORTH-Versionen runden immer in Richtung 0, so daß –10/3 –3 ergibt. Nachdem der Quotient mit diesen Regeln berechnet wurde, wird der Rest mit folgender Formel bestimmt:
Dividend = Divisor * Quotient + Rest
Multiplikation mit Division: */ hat einfache Länge mit Vorzeichen und folgende Auswirkung auf den Stapel:
n1,n2,n3——n1*n2/n3
Vergleichsoperationen: = < > und U< haben einfache Länge. < und > haben Vorzeichen, U< ist ohne Vorzeichen und = gilt für beides. D= D< und DU< haben doppelte Länge. D< hat Vorzeichen, DU keine Vorzeichen und D= beides. MAX und MIN zeigen die größere oder die kleinere von zwei Zahlen an. Sie haben einfache Länge mit Vorzeichen. DMAX und DMIN haben doppelte Länge mit Vorzeichen.
Absolute Werte: ABS und DABS gelten jeweils für einfache und doppelte Längen.
Bitweise Boolesche Operationen: AND, OR, XOR und NOT haben einfache Länge. Das Vorzeichen spielt keine Rolle.
Standardkombinationen: Einige Worte werden so oft gemeinsam eingesetzt, daß ihre Kombination ohne jeglichen Zwischenraum geschrieben wird und somit als einzelnes Wort gilt. So hat denn beispielsweise 1+ die gleiche Wirkung wie 1 +.
Darstellung: . und U. haben einfache Länge und können mit oder ohne Vorzeichen sein. D. hat doppelte Länge mit Vorzeichen.
Speicheradressierung: Eine Adresse mit einfacher Länge und ohne Vorzeichen kann sich auf die zwei Bytes dieser Adresse beziehen (wie mit @ und !) oder auf nur ein Byte (mit C@ und C! – C bedeutet „Character") oder auf die dort gespeicherte Vier-Byte-Zahl doppelter Länge (mit 2@ und 2!).

Neu definiert

Aus Gründen der Geschwindigkeit und Optimierung werden die Colon-Definitionen in FORTH „halb-compiliert". In diesem Artikel beschreiben wir die Compiliermethoden von FORTH und untersuchen, welche Wirkung die Neudefinition von bereits bestehenden Wörtern auf die Programmausführung hat.

Die halb-compilierten Colon-Definitionen werden zwar nicht in den Maschinencode übersetzt (wie bei einer vollständigen Compilierung), belegen aber wenig Platz und laufen schnell ab. FORTH Programme haben daher eine hohe Ausführgeschwindigkeit.

Es gibt zwar mehrere Methoden der Halb-Compilierung, doch wird die figFORTH-Methode an häufigsten eingesetzt. Bei der Definition des Wortes:

```
:SQUARE (n-n*n) DUP * ;
```

wird zunächst ein „Header" in das Vokabular eingetragen. Er enthält den Namen SQUARE und die Speicheradresse des Headers des zuletzt definierten Wortes. Da alle Header miteinander verbunden sind, kann der Computer das Vokabular vom zuletzt definierten Wort an rückwärts durchsuchen, bis er das gewünschte Wort findet. Die beiden Teile des Headers heißen „Namensfeld" und „Linkfeld".

Auf den Header folgt die Definition. Dort steht zunächst ein Code, der anzeigt, welche Art der Definition vorliegt: Es gibt unterschiedliche Codes für Colon-Definitionen, Variablen, Konstanten, mit CREATE angelegte Strukturen, Wörter, die in Maschinencode für das System definiert wurden und jede andere Art von Wörtern. Dieser Teil heißt „Codefeld".

Als nächstes kommt der Hauptteil der Colon-Definition. Er besteht aus den Adressen der Wörter, die von der Definition eingesetzt werden („Compilierungsadressen").

Hier die Definition von SQUARE:

Namensfeld: "SQUARE"
Linkfeld: Headeradresse des vorigen Wortes (2 Bytes)
Codefeld: Code für eine Colon-Definition (2 Bytes)
Parameterfeld: Compilierungsadresse von DUP (2 Bytes)
Compilierungsadresse von * (2 Bytes)
Compilierungsadresse von ; (2 Bytes)

Die Compilierung von Colon-Definitionen läuft folgendermaßen ab: Zunächst werden das Namensfeld und das Linkfeld angelegt. Bis hier ist der Vorgang für jede Definition gleich. Als nächstes setzt der Doppelpunkt (:) den speziellen Code für Colon-Definition ein. Von nun an wird jedes eingegebene Wort nicht auf normale Weise ausgeführt, sondern dessen Compilierungsadresse in das Vokabular eingetragen. Damit Sie erkennen können, daß dieser Arbeitsgang aktiviert ist, wird die Meldung OK unterdrückt.

Der Vorgang setzt sich so lange fort, bis ein (;) eingegeben wird. Das Semikolon hat außer seiner im Vokabular eingetragenen Funktion noch die Wirkung, FORTH sofort aus der Compilierung in den normalen Zustand zurückzubringen. Wörter, die als Teil des Compilierungsvorgangs ausgeführt werden, werden „unmittelbare" (immediate) Wörter genannt. Sie befinden sich normalerweise nicht im Vokabular, können aber dort speziell definiert werden. In diesem Fall wird im Vokabular jedoch nicht deren Compilierungsadresse eingetragen, sondern eine Art anonymer „Schatten", der die Abläufe vom „;" für die Programmausführung enthält.

Bei der Ausführung von SQUARE (beispielsweise bei der Eingabe von 4 SQUARE) sucht FORTH im Vokabular und findet dort die entsprechende Definition. Diese teilt FORTH mit, daß zuerst DUP, dann * und dann das Ende (der Definition) ausgeführt werden soll.

Da Zahlen keine Compilierungsadressen besitzen, werden sie in einem zusammengesetzten Format compiliert. Am Anfang steht dabei die Compilierungsadresse einer direkten Behandlungsroutine (Literal-Handler) – und dann die eigentliche Zahl. Ein Literal-Handler, der im Inneren einer Colon-Definition auftritt,

Codefelder

In Codefeldern ist die Codenummer die Speicheradresse des Maschinencodes, der bei Aufruf des Wortes ausgeführt wird. Jede Colon-Definition speichert die Adresse, zu der der FORTH-Interpreter nach Ausführung des Wortes zurückkehrt. Dafür wird die Rücksprungadresse auf einen speziellen Stapel – den „Rücksprungstapel" – geschoben und der Interpreter auf die Anfangsadresse der aktuellen Definition gesetzt.

Die Maschinencoderoutinen für Konstanten und Variablen, deren Parameterfelder nur eine einzige Zahl enthalten, sind anders aufgebaut: Bei sogenannten primitiven Wörtern wie + und * liegt der entsprechende Maschinencode direkt im Benutzerfeld, während das Codefeld die Adresse enthält.



Auch die Schneider-Computer CPC 464 und 664 gehören zu der wachsenden Zahl von Computern, die mit FORTH arbeiten können. Die Kuma-Version von FORTH entspricht dem figFORTH-Standard und bietet Fließkommaarithmetik, Möglichkeiten der Stringverarbeitung und Farbgrafik.

teilt dem System mit, daß die beiden folgenden Bytes keine Compilierungsadresse darstellen, sondern eine Zahl, die auf den Stapel geschoben werden soll.

Das nächste Problem tritt bei der Programmierung von Strukturen wie IF...THEN.... ELSE auf. Im Maschinencode oder in einfacheren BASIC-Versionen läßt sich diese Aufgabe mit Sprüngen lösen, wie:

```
1000 IF oberster Stapelwert = 0 THEN
     GOTO 1000
1010 REM folgende Zeilen ausführen, wenn
        die Bedingung wahr ist
     :
     :
1090 GOTO 1200
1100 REM folgende Zeilen ausführen, wenn
        die Bedingung falsch ist
     :
     :
1200 REM in beiden Fällen hier weiter-
        machen
```

Compiliertes FORTH funktioniert ähnlich. Doch obwohl FORTH die Wörter BRANCH (unbedingter Sprung) und ?Branch (obersten Stapelwert herunterziehen und springen, falls der Wert Null ist) besitzt, können sie nicht ohne weiteres verwandt werden, da sie sich nur in zusammengesetzter Form (wie eine compilierte Zahl) einsetzen lassen: Auf die Compilierungsadresse muß die Adresse folgen, auf die der Sprung stattfinden soll. Es ist Aufgabe der unmittelbaren (immediate) Wörter IF, ELSE, THEN etc., diese Sprungadressen herauszufinden und in die zusammengesetzten Formate einzufügen. So enthält IF beispielsweise die Compilierungsadresse von ?BRANCH und den Platz für die Sprungadresse, kann aber die Sprungadresse nicht einsetzen, da nicht bekannt ist, wo sich das ELSE befindet. IF legt statt dessen die Adresse auf den Stapel, und ELSE setzt später die Sprungadresse ein. Damit lassen sich Strukturen wie

```
IF...BEGIN...ELSE...UNTIL...THEN
```

abfangen. Sehen wir uns einige Beispiele an:

1) Nehmen Sie einmal an, Ihre FORTH-Version könnte zwar zwischen Groß- und Kleinschreibung unterscheiden, doch wären alle Standardwörter nur mit Großbuchstaben im Vokabular eingetragen, – das heißt, LOOP würde erkannt werden, loop dagegen nicht. Wenn Sie die Standardbefehle nun auch in Kleinbuchstaben eingeben wollten, könnten Sie definieren:

```
: drop DROP;
: roll ROLL;
: variable VARIABLE (auch das funktioniert);
```

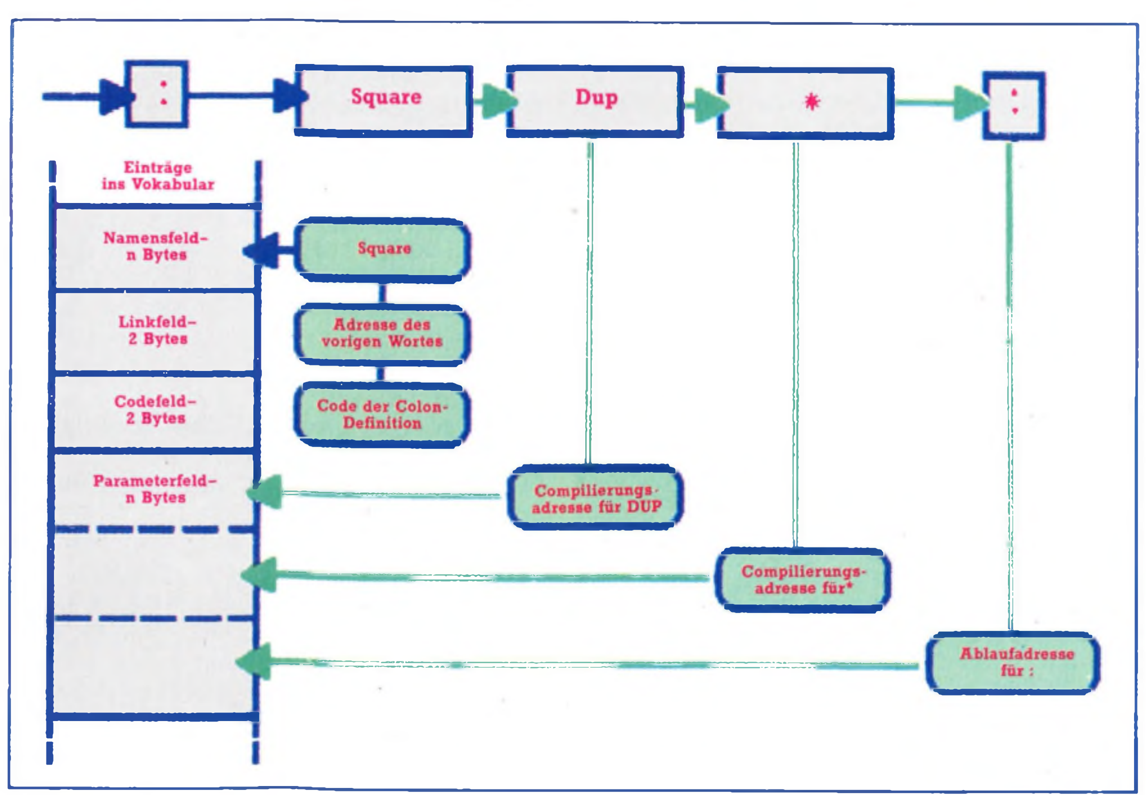
Schwierigkeiten gibt es mit

```
:loop LOOP (funktioniert nicht)
```

da loop ein unmittelbares Wort ist und daher schon bei der Definition ausgeführt wird. Die einzige Möglichkeit, LOOP mit Kleinbuchstaben zu definieren, ist:

```
:loop [COMPILE] LOOP ; IMMEDIATE
```

Hierbei müssen drei Situationen berücksichtigt werden. Zunächst die Definition: Während die Compilierungsadresse unter loop eingetragen wird, schaltet [COMPILE] die sofortige Ausführung ab, und IMMEDIATE kennzeichnet loop als unmittelbares Wort. Ferner ist zu berücksichtigen, daß loop während der Compilierung eines anderes Wortes eingesetzt und als unmittelbares Wort sofort ausgeführt wird. Da dies der Haupteinsatzbereich von loop ist, tritt dieser Fall häufig ein. Die dritte Situation



FORTH arbeitet mit der Methode der „Halb-Compilierung", um die Programmausführung zu beschleunigen. Bei Eingabe einer Colon-Definition werden im Vokabular unterschiedliche Felder angelegt, die mit dem Namensfeld, Linkfeld und dem Codefeld anfangen. FORTH schaltet dann den Compilierungsmodus (Rot) an, in dem das Wort nicht unmittelbar in das Vokabular eingetragen wird, sondern nur eine „Compilierungsadresse" im Parameterfeld der Definition. Nach dem ; schaltet FORTH wieder in den Ausführmodus (Schwarz) zurück.

Bedeutende Wörter

Mit [und] können Sie von einer Colon-Definition in den Ausführungsmodus schalten. LITERAL, ein unmittelbares (immediate) Wort, nimmt den obersten Wert des Stapels und compiliert ihn in einer Colon-Definition auf übliche Weise in eine Zahl. Normalerweise wird LITERAL zusammen mit [und] eingesetzt, um während der Compilierung einen Ausdruck zu berechnen, dessen Ergebnis compiliert werden soll. So hat

```
[ 9 16 * ] LITERAL
```

die gleiche Auswirkung wie die Zahl 144.
[COMPILE] ist ein unmittelbares Wort, das veranlaßt, daß das nächste Wort nicht ausgeführt, sondern direkt in die Definition eingeschlossen wird.
COMPILE ist kein unmittelbares Wort. Bei seiner Ausführung wird das darauffolgende Wort in das Vokabular eingeschlossen.
BRANCH und ?BRANCH werden wie in unserem Beispiel eingesetzt. >MARK und >RESOLVE geben bei einem Vorwärtssprung die Sprungadressen an: >MARK steht dabei an der Position des Sprunges und >RESOLVE an der Bestimmungsposition. <MARK und <RESOLVE werden auf die gleiche Weise für einen Rückwärtssprung verwandt, wobei <MARK vor der Bestimmungsposition steht und <RESOLVE an der Sprungposition.
IMMEDIATE wandelt das vorangehende Wort in ein unmittelbares Wort um.

entsteht beim eigentlichen Ablauf, wenn bei der Ausführung eines anderes Wortes die von LOOP compilierten Definitionen ablaufen.
2) Das Wortpaar ?DO und ?LOOP ähnelt DO und LOOP. Wenn dabei Anfangswert gleich Endwert ist, kommt die Schleife niemals zur Ausführung.

```
  ?DO entspricht OVER OVER > IF DO
?LOOP entspricht LOOP ELSE DROP DROP
    THEN
```

Bei den Definitionen

```
  : ?DO (hier ist der Fehler) OVER OVER > IF
    DO;
```

würde das gleiche Problem auftreten wie bei loop. Die korrekte Definition sieht folgendermaßen aus:

```
  :?DO
  COMPILE OVER
  COMPILE OVER
  COMPILE >
  [COMPILE] IF
  [COMPILE] DO
;IMMEDIATE

:?LOOP
  [COMPILE] LOOP
  [COMPILE] ELSE
  COMPILE DROP
  COMPILE DROP
  [COMPILE] THEN
;IMMEDIATE
```

3) Die Programmstruktur CASE bietet – abhängig von einem Steuerwert – eine ganze Reihe von Ablaufmöglichkeiten. CASE ist zwar nicht standardmäßig vorgesehen, läßt sich aber leicht definieren. Hier das Format:

```
  Wert CASE
1.Möglichkeit OF . . .ENDOF
2.Möglichkeit OF . . .ENDOF
3.Möglichkeit OF . . .ENDOF
falls keine der Möglichkeiten eintrifft
ENDCASE
```

Die Variable „Wert" kann beispielsweise der ASCII-Code einer Taste sein. Damit lassen sich per Tastendruck unterschiedliche Abläufe aufrufen.

In der Definition von CASE benötigt jedes OF einen bedingten Sprung auf ein ENDOF und jedes ENDOF einen Sprung auf ENDCASE, während ENDCASE unter Umständen mehrere Sprungadressen einfügen muß. Da die meisten FORTH-Versionen für diese Aufgabe keine Wörter besitzen, müssen Sie diese Sprünge mit >MARK und >RESOLVE selbst ausarbeiten. >MARK schiebt dabei (wie IF) die Adresse des für die Sprungadresse reservierten Platzes auf den Stapel, während >RESOLVE sie wieder herunterzieht und die Adresse einfügt.

```
  : CASE (Laufzeit: Wert — Wert)
          ( compilieren: —0)
          0 (Zahl der OFs — bisher keine)
; IMMEDIATE
: OF ( Laufzeit: Wert, falls möglich — [bei
      Übereinstimmung])
( Wert, falls möglich — Wert [keine Überein-
      stimmung])
(compilieren: — Markierung für den Sprung
      auf ENDOF)
  COMPILE OVER
COMPILE =
COMPILE ?BRANCH >MARK
COMPILE DROP (Wert löschen falls
    Übereinstimmung)
; IMMEDIATE
: ENDOF (Laufzeit: —)
  ( compilieren: OF Zähler,
        OF Markierungen —
( Markierung für den Sprung auf ENDCASE,
  OF Zähler + 1)
COMPILE BRANCH >MARK
SWAP >RESOLVE (Sprungadresse von OF
    einsetzen)
SWAP 1+
; IMMEDIATE
: ENDCASE (Laufzeit: Wert —
    (compilieren: Markierung der ENDOFs,
    OF
    Zähler —)
  COMPILE DROP
  0 ?DO
    > RESOLVE
    ?LOOP
    ; IMMEDIATE
```

Vor dem Einsatz sollten Sie die Abläufe mit einigen Testzahlen überprüfen.

FORTH-Arrays

FORTH verfügt zwar standardmäßig nicht über Arrays, kann diese Strukturen aber definieren. Wir demonstrieren diese Definitionsart.

Bei der Programmierung des Sieb des Eratosthenes haben wir mit CREATE bereits einige Konzepte gestreift, die bei Array-Definitionen eingesetzt werden. CREATE hatte dort die Funktion

```
CREATE BITS 8192 ALLOT
```

und enthielt einen (im Vokabular eingetragenen) Header namens BITS und ein Codefeld. Das Codefeld veranlaßte, daß BITS während der Laufzeit seine Parameterfeldadresse auf dem Stapel hinterließ. Da CREATE selbst jedoch kein Parameterfeld anlegt, zieht ALLOT eine Zahl vom Stapel und reserviert im Vokabular Platz für diese Anzahl Bytes. Der Platz liegt unmittelbar hinter dem Codefeld, das von CREATE eingeschlossen wird und dient BITS als Parameterfeld.

Hier ein einfaches Beispiel für die Definition des Wortes 3**, das eine dritte Potenz berechnet. Zum Speichern aller nur möglichen Ergebnisse verwenden wir ein Array, das mit CREATE und dem FORTH Wort „," (Komma) angelegt wird:

```
VARIABLE MAXINDEX
:POTENZ, (——)
  (schließt alle Dreierpotenzen ein, die in
  zwei Bytes passen und setzt den Index der
  höchsten Potenz in MAXINDEX.)
—1 MAXINDEX !
1 (kleinste Potenz)
BEGIN
  DUP,( Potenzzahl ins Vokabular aufnehmen )
  1 MAXINDEX +!
  3 UM* ( nächste Potenz, doppelte Länge )
    ( in FORTH-79 und figFORTH UM*
    durch U* ersetzen )
UNTIL (bis die nächste Potenz mehr als 2
    Bytes belegt )
DROP ( niederwertige Bytes der Potenz mit
    doppelter Länge )
;
CREATE 3POTENZ POWERS
    ( ALLOT wird nicht gebraucht, da das
    Komma das Einschließen erledigt )
:3** ( n —— 3**n )
MAXINDEX @ OVER U<IF
  DROP 0 (n liegt nicht zwischen 0 und
    MAXINDEX. Ergibt 0)
  ELSE ( der Speicherinhalt von 3POTENZ
    + 2*n wird gebraucht)
2 * 3POTENZ +@
THEN
```

3POTENZ ist ein eindimensionales numeri-

Array-Strukturen

Mit CREATE und DOES> lassen sich in FORTH leicht Arrays anlegen. Das Diagramm zeigt Schritt für Schritt, wie FORTH elf 1ARRAY RUNS ausführt. 1ARRAY wurde zuvor im Vokabular als Colon-Definition eingerichtet. Das Bild zeigt auch die Auswirkungen auf den Stapel.

Colon-Definition	Elf „1ARRAY RUNS“	Stapel
: 1ARRAY		
CREATE	CREATE teilt RUNS ein Namensfeld und ein Codefeld zu. RUNS hinterläßt beim Einsatz nun seine Pfa auf dem Stapel.	11
DUP +	DUP+ multipliziert 11 mit zwei und schiebt das Ergebnis auf den Stapel.	22
HERE	HERE schiebt die nächste verfügbare Adresse des Vokabulars auf den Stapel. Da wir gerade RUNS eingeschlossen haben, ist dies die Pfa von RUNS.	PFA / 22
OVER	OVER kopiert 22 oben auf den Stapel (direkt über die Pfa von RUNS).	22 / PFA / 22
ALLOT	ALLOT zieht 22 vom Stapel und reserviert im Vokabular 22 Bytes, die (da wir gerade RUNS dort eingesetzt haben) das Parameterfeld für RUNS bildet.	PFA / 22
SWAP 0 FILL	SWAP setzt 22 über die Pfa von RUNS, schiebt dann 0 auf den Stapel. FILL nimmt die drei Parameter und stellt alle 22 Bytes des Parameterfeldes auf Null.	0 / 22 / PFA → LEER
DOES>	Laufzeit (hinter DOES>)	
	99 2 RUNS ! setzt beispielsweise den Stapel wie im Bild gezeigt und speichert den Wert 99 im Element 2 des Arrays RUNS wie folgt:	PFA / 2 / 99
SWAP	SWAP setzt den Arrayindex (2) an die Stapelspitze.	2 / PFA / 99
DUP +	DUP+ verdoppelt den Wert ...	4 / PFA / 99
+	+ addiert den Index (der nun den Offset bildet) auf die Pfa von RUNS ...	PFA +OFF-SET / 99
;		
!	! nimmt 99 und speichert den Wert an der richtigen Adresse.	LEER

sches Array. Sie könnten es ebensogut mit ALLOT und ! anlegen. String-Arrays werden auf ähnliche Weise aufgebaut. Weil dabei jedes Zeichen aber immer nur von einem Byte dargestellt wird, müssen Sie C, , C! und C@ nehmen, statt , ,! und @.

Array-Einsatz

Der Einsatz eines Arrays wie 3POTENZ muß allerdings mit

```
2 * 3POTENZ +
```

gekennzeichnet werden, da 3POTENZ nicht weiß, daß es ein Array ist. Das Wort hinterläßt nur seine Parameterfeldadresse zur weiteren Bearbeitung. Mit CREATE (figFORTH: <BUILDS) und DOES> lassen sich jedoch „intelligentere" Wörter anlegen, CREATE und DOES> definieren neue Wörter, die eigentlich verbesserte Versionen von CREATE – und damit neue Definitionswörter sind. Sie enthalten nicht nur CREATE, sondern führen gleichzeitig auch andere Vorgänge aus. Die neuen Definitionswörter können beispielsweise darüber informieren, welche Aufgaben ein neu angelegtes Wort hat, statt nur die Parameterfeldadresse (pfa) auf den Stapel zu schieben.

```
:1ARRAY ( n —— )
  ( baut ein eindimensionales numerisches
  Array mit der Dimension n auf )
CREATE ( der Name wird beim Einsatz von der
  Position hinter 1ARRAY genommen )
DUP + (ein schneller Weg, 2 * auszuführen)
HERE (beachten Sie die Parameterfeld-
  adresse  für FILL)
OVER ALLOT
  ( jetzt 2*n, pfa)
SWAP 0 FILL
  ( alle Bytes des Parameterfeldes auf
    0 stellen )
  DOES> (Subscript, pfa —— Adresse des
  Elements )
SWAP DUP + (pfa, 2* subscript)
+
;
```

Das Beispiel besteht aus zwei Teilen: Die vor DOES> stehende Definition wird beim Einsatz von 1ARRAY ausgeführt. Sie legt das Parameterfeld des neuen Wortes an.

```
11 1ARRAY RUNS
```

richtet das neue Wort RUNS ein und ordnet ihm mit ALLOT ein aus 22 Bytes bestehendes, und mit Nullen initialisiertes Parameterfeld zu. Der zweite Teil von 1ARRAY (nach DOES>) wird beim Ablauf aktiviert. RUNS schiebt dabei zwar auch hier noch seine Parameterfeldadresse ganz einfach auf den Stapel, setzt dann aber seine „Intelligenz" ein und legt an dieser Stelle elf Variablen mit den Namen:

```
0 RUNS
1 RUNS
:

:
10 RUNS
```

an, die sich wie normale Variablen einsetzen lassen. Hier ein Beispiel:

```
99 2 RUNS !
4 RUNS @.
```

Dieser Einsatz von CREATE. . .DOES> war recht einfach. Mit etwas mehr Erfahrung können Sie jedoch weitaus komplexere Aufgaben damit erledigen. Bei der Entwicklung von FORTH-Programmen wird in Variablendefinitionen oft das @ vergessen. Mit CREATE und DOES> lassen sich nun Variablen definieren, die ohne @ auskommen. Normalerweise wird dabei nur der Variablenwert (wie bei einer Konstanten) auf dem Stapel zurückgelassen. Wenn Sie den Wert jedoch ändern wollen, entsteht ein Problem, das Sie am besten mit dem Wort → lösen. Hier der Ablauf:

Neuer Wert → Variablenname im neuen Stil

→ hat nur die Aufgabe, dem System (durch Setzen einer Variablen) mitzuteilen, daß dieses Wort verwendet wurde. Die Variable „im neuen Stil" weiß dadurch jedoch, daß sie einen neuen Wert vom Stack ziehen muß, statt ihren aktuellen Wert auf den Stapel zu schieben.

```
VARIABLE → USED
0 → USED !
: → — 1 → USED!;
```

Wir wollen nun für das Definitionswort VARIABLE ein neues definieren. Das neue Wort ist ebenfalls ein Definitionswort und arbeitet mit

```
CREATE. . .DOES>.
  : VAR ( — — )
    CREATE
    0, (teilt 2 Bytes zu, die mit 0 initialisiert sind )
  DOES >
    → USED @ IF ( neuer Wert, pfa — — )
    ! (neuen Wert setzen )
    0 → USED!
  ELSE (pfa — — aktueller Wert )
  @
  THEN
  ;
```

Hier die Definition der VAR Variablen x. Sie wird auf 99 gesetzt und der Wert angezeigt:

```
VARx
99 → x
x.
```

CREATE und DOES> lassen sich außerordentlich flexibel einsetzen, da sie zwei Eigenschaften miteinander kombinieren, die normalerweise separat arbeiten. Die meisten Program-

miersprachen enthalten „passive" Daten, die bearbeitet werden und „aktive" Routinen, die die Arbeit ausführen und mit Daten gefüttert werden. Mit CREATE und DOES> aber können Sie intelligente Daten anlegen, die sich während des Programmablaufs selbst bearbeiten. Wenn Sie dieses flexible Konzept einsetzen, müssen Sie jedoch die Trennung zwischen Daten und Programmen vergessen, die bei den meisten anderen Programmiersprachen verlangt wird.

Nützliche Wörter

,(n — —) trägt n mit zwei Bytes im Vokabular ein. Die Ein-Byte Version ist C, .
HERE (— — Adresse) schiebt die Adresse des im Vokabular reservierten Platzes auf den Stapel und kennzeichnet damit die Stelle, an der der nächste Eintrag stattfinden soll.
FILL (Adresse, n, Füllelement — —) füllt von der angegebenen Adresse n Bytes mit dem Füllelement aus.

Der Array-Aufbau

Die mit 1ARRAY definierten Arrays arbeiten schnell, prüfen aber nicht, ob der angegebene Index überhaupt zulässig ist und können so durchaus Werte außerhalb des Arrays überschreiben. Wenn bei 1ARRAY die Dimension am Anfang des Parameterfeldes gespeichert wird, kann der Ablaufteil prüfen, ob ein angegebener Index kleiner ist als die Dimensionierung und eine Fehlermeldung ausgeben, falls dies nicht der Fall ist.

Einige Sprachen wie PASCAL und BASIC überprüfen Arrayindizes automatisch, während andere (beispielsweise C) für hohe Geschwindigkeit ausgelegt sind und keinen Test ausführen. In FORTH können Sie selbst entscheiden, welchen Weg Sie gehen wollen und hierzu dann die Sprache entsprechend programmieren.

Weiterhin setzen einige Programmiersprachen den Index 0 als erstes Element ein, andere dagegen das Element 1 (oder ein anderes). Auch hier läßt Ihnen FORTH weiterhin uneingeschränkt die freie Wahl.

Mehrfachdimensionale Arrays arbeiten nach dem gleichen Prinzip, sind aber komplizierter aufgebaut. Ein mehrfachdimensionales Array stellt man sich normalerweise als dreidimensionalen Block vor. Der Computer speichert ihn Zeile für Zeile in einer langen Liste. Sie können die Indizes als Offsets für diese Liste einsetzen, indem Sie diese mit den Dimensionen multiplizieren.

Für ein dreidimensionales Array mit den Dimensionen 2, 3 und 4 (insgesamt 24 Elemente) lassen sich die Indizes i, j und k folgendermaßen in Offsets umwandeln:

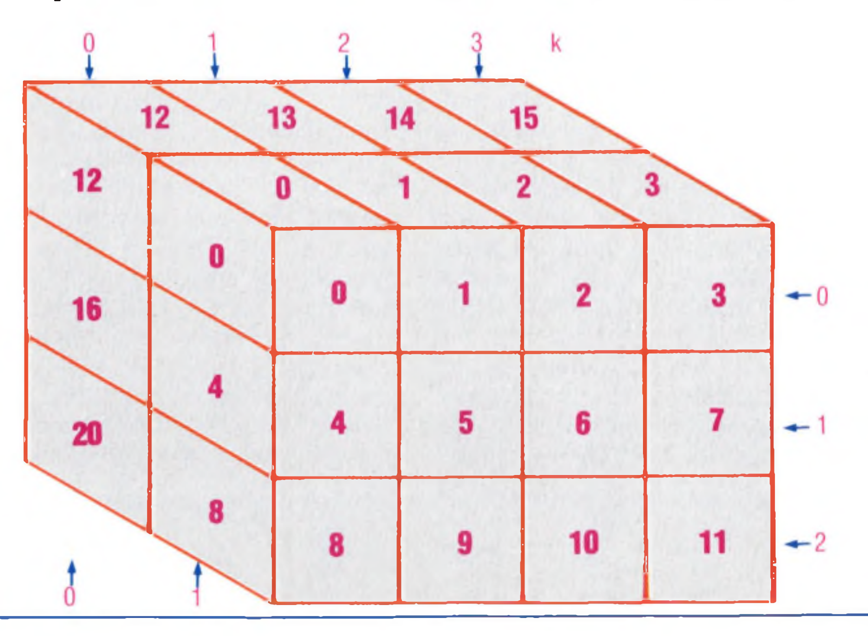
```
(i*3 + j) * 4 + k
```

Hier die Definition von 3ARRAY:

```
: 3ARRAY (d, e, f — —)
  ( legt ein dreidimensionales Array mit den
  Dimensionen d, e und f an)
CREATE
  OVER, DUP, ( e und f eintragen )
  HERE ( d, e, f, pfa+4 )
  2 ROT * ROT * ROT * ( pfa+4, 2*d*e*f )
  DUP ALLOT ( 2*d+e+f Bytes eintragen)
  0 FILL ( mit 0 initialisieren )
DOES> ( i, j, k, pfa — — Adresse )
  DUP@ ( i, j, k, pfa, e )
  4 ROLL * ( j, k, pfa, i*e )
  3 ROLL + ( k, pfa, i*e+j )
  OVER 2+@ ( k, pfa, i*e+j, f )
  * ( k, pfa, [i*e+j]* f )
  ROT + DUP + ( pfa, 2*[[i*e+j]*f+k] )
  SWAP 4 + +
;
```

(Beachten Sie, daß die Zahl vor ROLL in FORTH-79 um eins höher sein muß.)

Nach dieser Methode lassen sich ähnliche Arrays (4ARRAY, 5ARRAY, etc.) anlegen. Es kann sogar ein Wort definiert werden, das eine Zahl vom Stapel zieht und Ihnen mitteilt, wie viele Dimensionen sich auf dem Stapel befinden. Falls Ihnen dies zu kompliziert erscheint, sollten Sie nachlesen, wie die Programmiersprache C Arrays anlegt.



Wortspiele

Mit der Verarbeitung von Strings beschließen wir unsere FORTH-Serie. Dabei simulieren wir verschiedene Möglichkeiten der BASIC-Stringverarbeitung in FORTH.

FORTH kann Strings auf drei verschiedene Weisen im Speicher unterbringen. Jede Methode arbeitet mit einer Adresse, die sich leicht einsetzen läßt und auf den Speicherbereich zeigt, in dem der String untergebracht wurde. Alle drei Methoden verwenden jedoch unterschiedliche Techniken, um die Länge des Strings festzustellen. Beim „gezählten String" wird die Längeninformation im ersten Byte des Bereichs untergebracht, in dem auch der String gespeichert wurde. Der „Textstring" legt den Längenwert auf den Stapel, wo er verändert werden kann. Einige FORTH-Versionen unterstützen Strings, die mit NUL beendet sind. Dabei ist das Ende des Strings einfach durch ein Nullbyte markiert.

Wir hatten bereits gesehen, daß sich das Wort ." zwar leicht einsetzen läßt, für kompliziertere Stringverarbeitung aber nicht geeignet ist, da dabei zwei wesentliche Probleme entstehen: Wo soll der String untergebracht werden und wie läßt er sich speichern? Da Abläufe dieser Art recht komplex sind, lohnt es sich, Wörter zu definieren, die die Einzelheiten für Sie erledigen.

Strings können in zwei Bereichen untergebracht werden – in den mit ALLOT reservierten Parameterfeldern der im Vokabular eingetragenen Wörter und in einem Bereich namens „Pad". Pad ist ein Zwischenspeicher, den FORTH auch für andere Aufgaben einsetzt. Bei jedem Vokabulareintrag oder jeder Interpretation eines FORTH-Wortes per Tastatur kann Pad überschrieben oder verschoben werden.

FORTH speichert Strings auf zwei Arten (als ASCII-Werte), die sich nur durch die Methode unterscheiden, mit der sie die Länge des Strings angeben.

Bei „gezählten Strings" wird die Länge (maximal 255 Bytes) in einem einzigen Byte unmittelbar vor den Zeichen gespeichert. Strings dieser Art lassen sich über die Adresse dieses Zählerbytes ansprechen. Zwar können Sie den String nicht auf den Stapel schieben, doch genügt es, seine Adresse dort abzulegen. Bei „Textstrings" wird die Länge nicht automatisch mit abgespeichert. Hier müssen Sie zwei Informationen angeben: die Adresse des ersten Zeichens und die Länge des Strings.

Der in Stringvariablen zur Verfügung stehende Platz ist begrenzt, da er im Parameterfeld liegt, dessen Länge mit ALLOT fest definiert wurde. Bei $VARIABLE müssen Sie daher nicht nur einen Anfangswert angeben, sondern auch die maximale Länge zukünftiger Zuordnungen. Dabei kann es vorkommen, daß die Länge der ersten Zuordnung für später eingegebene Strings nicht ausreicht.

Sehr praktisch ist das Wort ASCII. Es findet sich in den meisten FORTH-Versionen, gehört aber nicht zum Standard. ASCII schiebt den ASCII-Code des nachfolgenden Zeichens auf den Stapel. Bei einer Colon-Definition compiliert es für die angegebene Zahl ein numerisches Zeichen, das dann während der Laufzeit auf den Stapel geschoben wird.

```
32 CONSTANT BL ( ASCII-Code für ein
   Leerzeichen )
:ASCII ( —— ASCII-Code ) ( Zeichen im
   ASCII-Format )
   BL WORD ( nächstes Wort holen )
   1+ C@ ( und den ASCII-Code für sein
      erstes Zeichen holen )
   STATE @ IF ( IF gilt für die Colon-
      Definition )
      [ COMPILE ] LITERAL ( LITERAL ist
      unmittelbares Wort )
   THEN
;IMMEDIATE ( ASCII hat spezielle
   Compiliereigenschaften )
:$ VARIABLE ( Länge —— ) ( eingesetzt im
      Format: )
      ( Länge $VARIABLE Namen-
            initialisierer " )
   ( Im Parameterfeld wird 1 Byte für die )
   ( maximale Länge gefolgt von einem
   gezählten )
   ( String dieser Länge eingerichtet )
CREATE
255 MIN ( sicherstellen, daß der String
            nicht zu lang ist )
1 ALLOT ( für maximale Länge )
ASCII "WORD ( Länge, Initialisierer als
      gezählter String )
```

Gezählter String

Adresse
Stapel
8 E T H E L R E D
String-länge
Zeichen des Strings

Textstring

Länge
Adresse
Stapel
E T H E L R E D
Zeichen des Strings

Mit NUL beendeter String

Adresse
Stapel
E T H E L R E D 0
Ein NUL-Byte ist die Endmarkierung

```
    DUP C @ ROT MAX ( Initialisierer, Max.
          Länge )
    DUP HERE 1— C! ( max. Länge
          einsetzen)
    HERE SWAP 1+ ALLOT ( Initialisierer,
          Addresse zur Speicherung )
    OVER C@ 1+ CMOVE ( Initialisierer ein-
          kopieren )
  DOES > (pfa —— pfa + 1 ) ( Wert als ge-
          zählten String auf Stapel schieben )
    1 +
```

Die folgenden Zeilen definieren $@ und $! für $VARIABLES. Der Stapel wird im Text-String-Format (Adresse und Länge) eingesetzt, um die Werte ansprechen zu können.

```
:$@ ( pfa + 1 für die Stringvariable ——
      Text-String )
  COUNT
;
:$! ( Text-String, pfa + 1 für die String
      Variable —— )
  ( kürzt den Wert, falls er für die Variable
      zu lang ist )
  DUP 1 — C@ ROT ( adr, pfa, max. Länge,
                   wirkliche Länge )
  MIN            ( adr, pfa+1, einge-
                   setzte Länge )
  OVER OVER SWAP ( addr, pfa+1, Länge,
                   Länge, pfa+1 )
  C!             ( neue Länge spei-
                   chern )
  SWAP 1 + SWAP ( adr, pfa+2, Länge )
  CMOVE
;
;
```

Sie können nun Stringvariablen anlegen wie:

```
0 $VARIABLE ZIFFERN 0123456789"
```

(Die 0 wird automatisch auf 10 gesetzt, um die 10 vorhandenen Zeichen zu berücksichtigen. Danach ist die maximale Länge der Stringvariablen auf 10 festgelegt.)

```
ZIFFERN $ @ TYPE
```

(zeigt 0123456789 an)

```
255 $ VARIABLE XPAD XXX"
```

(ein stabilerer Pad für 255 Zeichen.)
Es wäre praktisch, wenn sich Stringwerte per Tastatur verändern ließen. Die Wörter " und $INPUT entsprechen den BASIC-Anweisungen LET A$ = „. . ." und INPUT A$.

```
:"      ( —— Adresse, Länge des Text-
String )
        ( außerhalb der Colon-Definition im
        Format — "text" eingesetzt )
ASCII "WORD COUNT ( Text-String holen )
XPAD $!              ( in XPAD speichern )
XPAD $@              ( Adresse und Länge
;                  bleiben unverändert)
```

Das Wort " erhält seinen String auf die gleiche Weise, wie bei Wörtern vom Typ VARIABLE ein neuer Name definiert wird. Diese Art Wörter können daher in Colon-Definitionen nicht funktionieren, lassen sich aber so einsetzen:

```
"Papagei" PET $! PET $@ TYPE
"Pinguin" PET $! PET $@ TYPE
```

Hier ein Wort, das Eingabebefehle imitiert:

```
:$INPUT ( pfa+1 der String Variable —— )
        ( wartet darauf, daß Sie einen
        String eingeben und kopiert ihn in
        die Variable)
DUP DUP 1+    ( pfa+1, pfa+1, pfa+2 )
SWAP 1—C@ 2—    ( pfa+1, pfa+2,
max.Länge—2 )
EXPECT
SPAN @ SWAP C!
;
```

Das Programm informiert EXPECT, nicht mehr Zeichen zu holen, als Platz in der Variablen ist. Bei einer bestimmten FORTH-Version kann hinter dem String zusätzlich ein NUL-Zeichen eingesetzt werden. Zur Sicherheit wird von der maximalen Länge daher 2 abgezogen, da sonst im Vokabular der Kopfsatz des nächsten Wortes zerstört werden könnte.

In FORTH-79 und figFORTH schreibt EXPECT in jedem Fall ein oder zwei NUL. Diese Zeichen sind notwendig, da die älteren FORTH-Versionen kein SPAN enthalten. Dabei muß ein zusätzliches Wort die Zeichen bis NUL zählen und so die Länge des Strings feststellen.

Jetzt ist es möglich, das BASIC-Programm

```
10 PRINT "Wie heißt Du?"
20 INPUT A$
30 PRINT "Nett, Dich kennen-
   zulernen,";A$; "."
40 PRINT "Bis zum nächsten Mal! Tschüß!"
```

mit selbstdefinierten Befehlen auch in FORTH zu schreiben:

```
30 $VARIABLE NAME
:HALLO!
   ." Hallo! Ich bin der Computer. Wie heißt
     Du?" CR
   NAME $INPUT
   ." Das ist aber ein interessanter Name,"
     NAME
   $@ TYPE
   "." ( Punkt anzeigen ) CR
   ." Paß auf Dich auf, OK? Schönen Tag
       noch!"
   CR
;
```

Mit BASIC lassen sich schnell kurze Programme schreiben, denen alle Fähigkeiten der Sprache (Fließkomma, Arrays, Strings etc.) zur Verfügung stehen. Für längere Programme eignet sich BASIC jedoch nicht, da es immer an die niedrige Struktur der Sprache gebunden ist – z. B. die Zeilennummern. Sehen Sie sich im Vergleich einmal unser FORTH-Wort HALLO! an. Es ist einfach aufgebaut und gibt ein ungefähres Verständnis davon, wie $INPUT, $ @ und ähnliche Wörter arbeiten. Die genaue Funktion läßt sich immer über die darin eingesetzten Wörter feststellen. Für den Programmablauf brauchen Sie jedoch nur zu wissen, was ein Wort ausführt, d. h. welche Werte auf dem Stapel sein müssen und welche Ergebnisse dort abgelegt werden.

Geflügelte Worte

STATE (-- Adresse) ist eine Variable, die unmittelbare Wörter verstehen können. Wenn ihr Wert auf wahr (nicht-null) steht, werden sie bei der Compilierung einer Colon-Definition aktiviert.
WORD (End-Zeichen -- gezählter String) liest das nächste Wort aus dem Eingabepuffer der Tastatur. Auf diese Weise erhält CREATE beispielsweise die Namen neuer Befehle. Das End-Zeichen ist normalerweise 32 (Leerzeichen). Es können aber auch andere End-Zeichen eingesetzt werden, wie * in unserer bekannten Definition von $VARIABLE.
EXPECT (Adresse, max. Zeichenzahl --) wird für neue Tastatureingaben eingesetzt. Die Zeichen werden bis zu dem angegebenen Maximum eingelesen und an der angegebenen Adresse abgelegt.
SPAN (-- Adresse) Diese Variable wird von EXPECT gesetzt und zeigt an, wieviele Zeichen eingelesen wurden. Nur FORTH-83 verfügt über SPAN. Ältere FORTH-Versionen markieren das Ende mit NUL-Zeichen.
PAD (-- Adresse des Pad) CMOVE (Ursprungsadresse, Zieladresse, Zeichenzahl --) kopiert die angegebene Anzahl Bytes von der Ursprungsadresse an auf die Zieladresse im Speicher und wird hauptsächlich für Stringkopien eingesetzt. Hier ist jedoch Vorsicht geboten, wenn der Zielbereich hinter dem Ursprungsbereich liegt und beide sich überlappen.
CMOVE> (Ursprungsadresse, Zieladresse, Zeichenzahl --) Dieses Wort gibt es nur in FORTH-83. Es löst das Problem von CMOVE, indem es die Bytes in einer anderen Reihenfolge kopiert.
COUNT (gezählter String -- Adresse, Länge des Text-Strings) Der String bleibt der gleiche, doch das Format wandelt sich vom gezählten String zum Text-String.

Fachwörter von A bis Z

Hier werden einzelne Fachausdrücke eingehend behandelt. Da bei der Kommunikation mit dem Computer meist die englische Sprache verwendet wird, werden hier zunächst die englischen Begriffe genannt, dann die deutsche Übersetzung. In den Gesamtindex werden sowohl deutsche als auch englische Stichwörter aufgenommen, damit Sie es leichter haben, das von Ihnen Gesuchte zu finden.

FORTH = FORTH
Die Sprache FORTH wurde 1972 von dem Astronomen Charles Moore entwickelt, nachdem er feststellen mußte, daß FORTRAN für seine Teleskop-Steuerprogramme nicht ausreichend war. Die Schwierigkeiten rührten daher, daß Struktur und Verarbeitungsformen bei FORTRAN zu sehr an der mathematischen Lösung wissenschaftlicher Probleme orientiert sind. FORTH bedient sich eines Lexikons von „Primitives“ (Stammwörtern) für die elementaren Funktionen und eines Editor/Compiler/Interpreter-Systems.

Der Editor ermöglicht die Definition neuer (unterprogrammähnlicher) Funktionen anhand des vorhandenen Wortschatzes. Die Namen werden dem Lexikon als neue „Wörter“ zugefügt. Die Funktionen werden compiliert und gespeichert und sind über den Interpreter jederzeit durch Eingabe des betreffenden Wortes ausführbar. Den RAM-Bereich behandelt FORTH als einen einzigen großen LIFO(Last In First Out)-Stack – den Programmspeicher dagegen als Gebilde aus unabhängigen Abschnitten. Jede Funktion verfügt über eine eigene Unterprogrammadresse. Das Programm springt zur Ausführung auf diese Adresse, greift alle nötigen Parameter vom LIFO-Stack, stapelt die Resultate dorthin zurück und geht zu der nächsten Adresse über.

Beim Programmieren in FORTH kann man für jede Anwendung einen maßgeschneiderten Befehlssatz entwickeln. Die Hauptvorteile sind die Anpassungsfähigkeit und die Geschwindigkeit der Sprache. Der in der Abbildung gezeigte Jupiter Ace wurde mit FORTH als systemeigener Sprache ausgeliefert.



FORTRAN = FORTRAN
FORTRAN (FORmula TRANslator = Formelübersetzer) wurde 1956 als erste kommerzielle Hochsprache bei IBM entwickelt. IBM bewies damit, daß auch ein dem gewohnten Englisch nahestehender Programmiercode schnell und effizient compilierbar sein kann. Die Sprache machte den Rechnereinsatz vor allem für Ingenieure und andere Wissenschaftler interessant. Sie waren eher bereit, etwas dem vertrauten mathematischen Formalismus Entsprechendes zu erlernen, als die Zeit und Geduld für das Programmieren im Maschinencode aufzubringen. FORTRAN wurde ein durchschlagender Erfolg und ist auch heute noch die verbreitetste Hochsprache.

Als große Errungenschaft galt anfangs die Möglichkeit, Bibliotheken mit unabhängig compilierten FORTRAN-Unterprogrammen aufzubauen. Alle Großrechner verfügen über derartige Programmbibliotheken. Wegen ihrer Bedeutung als „Quellen“ wurden andere Sprachen – zum Beispiel PASCAL – so ausgelegt, daß sie auf FORTRAN-Bibliotheksprogramme zugreifen können.

FORTRAN hat eine stolze Zahl von Abkömmlingen, wie ALGOL, PASCAL und BASIC. Das historische Gewicht ist jedoch darin zu sehen, daß FORTRAN den Computer aus den Rechnerlabors der Universitäten herausholte und ihn in Ausbildungsstätten und Fabrikhallen zu einem selbstverständlichen Werkzeug von Wissenschaft und Technik werden ließ. Der Büro- und der Heimcomputer lagen dann nicht mehr fern: FORTRAN machte den Rechner auch dem Nicht-Fachmann zugänglich und leitete einen wichtigen Schritt auf dem Weg zu einer benutzerfreundlichen Datenverarbeitung ein.

Fourth Generation = Vierte Generation
In der Rechnerentwicklung umfaßt eine Generation immer etwa ein Jahrzehnt. Es beginnt jeweils mit dem Aufkommen einer aufwendigen neuen Technologie, die mit Ende dieser Zeitspanne alltäglich wird. Die Anfänge der „ersten Generation“ lagen vor 1950 bei den ersten speicherprogrammierten Rechnern mit Elektronenröhren. Die „zweite Generation“ begann Ende der 50er Jahre mit der diskreten Transistorlogik. Die „dritte Generation“ vom Anfang der 60er Jahre (typischer Vertreter ist der IBM 360) ist durch die Verwendung integrierter Schaltungen und die Entwicklung umfänglicher Betriebssysteme gekennzeichnet. Und die „vierte Generation“ wurde zu Beginn der 70er Jahre mit der Einführung der LSI- und der VLSI-Technik eingeleitet. Diese Schaltungstechnik wird heute bei Großrechnern wie bei Microcomputern angewandt. Auch die Micros haben einige Entwicklungsstadien durchgemacht, bis sie den heutigen Stand als vollwertige Rechner mit beachtlicher Speicherkapazität, integrierter Software und der Fähigkeit zu Mehrprogramm- sowie Verbundnetz-Betrieb erreichten.

Bildnachweise
1233, 1258: Ian McKinnell
1234, 1253, 1256, 1257: Kevin Jones
1236, 1237, 1239, 1240, 1250: Liz Dixon
1238: Wattleworth Silk McConachie, Chris Tucker
1245, 1246, 1247: Chris Stevens
1252: Kevin Jones, Liz Dixon
1254: Liz Heaney

Fachwörter von A bis Z

Source Code = Quellprogramm
Ein Quellprogramm ist in einer Compiler- oder Assemblersprache geschrieben. Daraus muß vor Ablauf im Rechner durch Übersetzen in den Maschinencode ein „Objektprogramm" erzeugt werden. Das Quellprogramm, z. B. in PASCAL oder COBOL, wird zunächst einer syntaktischen Prüfung unterzogen; treten dabei keine Fehler auf, folgt die Compilierung. Auch ein Assemblerprogramm mit seinen mnemotechnischen Befehlen und symbolischen Adressen stellt ein Quellprogramm dar, das durch Assemblieren in die Maschinensprache umgesetzt wird.

Split Screen = Fenstertechnik
Bei dieser Technik wird der Bildschirm in zwei oder mehr Felder unterteilt, so daß oben auf dem Schirm ein Menü stehen kann, während darunter der Text dargestellt wird. Die untere Schirmhälfte läßt sich dabei unabhängig von der oberen bearbeiten, und der Benutzer kann auch meist beliebig zwischen den Fenstern hin und her springen. Das ist beispielsweise beim Editieren langer Schriftstücke sehr praktisch, weil ältere Passagen parallel zum Neuentwurf überarbeitet werden können. Besonders bei Abenteuerspielen finden Sie jetzt immer häufiger auch einen geteilten Grafikschirm.

Spooler = Spooler
Systemprogramme, die den Datenfluß an Peripheriegeräte (meist Drucker oder externe Speicher) auffangen und nach dem Warteschlangenprinzip in eigener Regie verwalten, heißen „Spooler". Sie sind vor allem deshalb zweckmäßig, weil die Ausgabegeschwindigkeit des Prozessors sehr viel größer ist als die Aufnahmefähigkeit der Peripherie. Wenn Daten dann zunächst in einen Puffer gesteckt werden, braucht die CPU nicht auf das externe Gerät zu warten. Zumeist arbeitet der Spooler unabhängig von der CPU und läßt sich direkt vom zugehörigen Peripheriegerät mitteilen, wann wieder neue Daten gesendet werden können.

Hier werden einzelne Fachausdrücke eingehend behandelt. Da bei der Kommunikation mit dem Computer meist die englische Sprache verwendet wird, werden hier zunächst die englischen Begriffe genannt, dann die deutsche Übersetzung. In den Gesamtindex werden sowohl deutsche als auch englische Stichwörter aufgenommen, damit Sie es leichter haben, das von Ihnen Gesuchte zu finden.

15
8
255
197

„Stacks" oder Stapelspeicher spielen in der Datenverarbeitung für das kurzfristige Deponieren von Informationen eine wichtige Rolle. Sie sind besonders einfach zu handhaben. Beim LIFO-Stack ist wie beim Würfelstapel aus dem Baukasten der Zugriff nur von oben her möglich. Die Reihenfolge der gespeicherten Größen bleibt vom Eintragen bis zum Auslesen unverändert, was für die Verwaltung der Rücksprungadressen bei verschachtelten Unterprogrammaufrufen ideal ist. Eine einfache Einsatzmethode für nicht so routinierte Anwender.

Square Wave = Rechtecksignal
Ein elektrisches Signal, das periodisch zwischen zwei Spannungspegeln hin- und herspringt, wird als „Rechtecksignal" bezeichnet. Eine solche Impulsfolge ist als Muster von binären Nullen und Einsen interpretierbar und dient zur Informationsübertragung, beispielsweise zwischen CPU und der Peripherie.

Stack = Stapelspeicher
Der „Stack" ist als spezieller Speicherbereich der CPU eines Computers zugeordnet und wird zur kurzfristigen Ablage von Registerinhalten und anderen Informationen benutzt. Die Kapazität des Stack und seine Lage im Arbeitsspeicher hängt vom jeweiligen Prozessor ab: Beim 6502 liegt der Stack auf der Nullseite (Zero Page) und faßt nur 256 Byte, beim Z80 dagegen können ihm beliebige Adressen und bis zu 64 KByte zugeteilt sein. Der Stack ist aber stets in gleicher Weise organisiert und geordnet.

Jeder Prozessor verfügt über besondere Stack-Befehle zum Eintragen (PUSH) und Entnehmen (POP) von Information. Beim üblichen LIFO-Stack werden die Daten „aufeinander" gestapelt; die letzten Einträge werden zuerst ausgelesen.

An diesem Prinzip kommt man beim Umgang mit dem Stack kaum vorbei: Die tieferstehende Information ist nur zugänglich, wenn der Stack von oben her weit genug abgearbeitet ist. Für den Zugriff hält ein bestimmtes CPU-Register, der „Stack Pointer" (Stapelzeiger), stets als aktuelle Adresse die der zuletzt angesprochenen Speicherzelle bereit. Der Stapelzeiger wandert beim Füllen und Leeren des Stack automatisch auf- bzw. abwärts, kann aber seitens des Programmierers bei Bedarf auch verschoben werden. Stack-Manipulationen sind nur im Maschinencode vorgesehen, aber es gibt sie auch in FORTH.

Bildnachweise
2213: Kevin Jones
2214: Liz Heany
2215–2217, 2240, U3: Caroline Clayton



Die Qual der Wahl

Für die meisten Hobby-Anwender reicht BASIC zur Programmentwicklung völlig aus. In Sonderfällen braucht man jedoch besser angepaßte Sprachen. Eine gründliche Analyse des Vorhabens sollte die Qual der Wahl jedoch lindern können.

Bis vor kurzem stand dem Heimcomputer-Freund ausschließlich BASIC oder Maschinensprache zur Verfügung. Heute sind dagegen fast alle verbreiteten Sprachen auch für die kleinsten Rechner erhältlich, die gesamte Palette steht beim IBM-Kompatiblen oder anderen MS-DOS-Rechnern zur Verfügung. Compiler und Interpreter können recht teuer sein, daher arbeiten nur wenige Anwender mit mehr als drei Sprachen.

Effektivität in bezug auf Speichergröße und die Arbeitsgeschwindigkeit sind die wichtigsten Faktoren. Das gilt besonders bei Programmen, die mit der Außenwelt kommunizieren sollen. Manche Programme wären in BASIC so schwierig zu erstellen, daß es leichter ist, eine neue, besser geeignete Sprache zu lernen und das Programm damit zu entwickeln. Ein Pro-

Sprache: BASIC
Vorteile: Leicht erlernbar und einfach in der Anwendung, weit verbreitet. Preiswert. Vielseitige Arithmetik-Funktionen. Komfortable Stringverarbeitung.
Nachteile: In vielen Versionen kaum Möglichkeiten für Modulprogrammierung und Steuerstrukturen. Keine Norm. Programmausführung langsam. Schlechtes File-Handling. Eingeschränkte Zahl von Datentypen und -strukturen. Programme undurchsichtig.
Anwendung: Kurze, einfache Programme mit Berechnungen oder Stringverarbeitung.

Sprache: LOGO
Vorteile: Solide mathematische Grundlage, leicht zu lernen. Turtle-Grafik. Gute Möglichkeiten zur Listenverarbeitung und für modulares Programmieren, viele Datentypen und Strukturen. Preiswert.
Nachteile: Auf höherer Ebene ist die Programmierung schwierig. Viele nicht-kompatible Versionen. Ausführung langsam.
Anwendung: Grafik, Listenverarbeitung, Lernen von Mathematik und Einführung in höhere Programmkonzepte.

Sprache: PASCAL
Vorteile: Gut strukturiert. Viele Datentypen. Weitgehend standardisiert. Verbreitet. Einfache Programmierung.
Nachteile: Ein/Ausgabe nicht klar definiert, File-Handling unbefriedigend.
Anwendung: Einüben guter Programmiertechnik. Allgemeine Aufgaben von beschränktem Umfang.

Sprache: FORTH
Vorteile: Schnelle Verarbeitung, vom Anwen-

gramm zur on-line-Verarbeitung von Lagerdateien wäre in COBOL durch den Einsatz indexierter Dateien sehr einfach zu erstellen. In BASIC müßte dagegen mit sequentiellen Dateien oder einem Hashing-Algorithmus und Random Access Files gearbeitet werden.

Bedienerfreundliche, leicht erlernbare Sprachen bereiten bei größerem Programmumfang oft Probleme. Hier kommt es darauf an, daß sich das Programm in selbständige Module zerlegen läßt, die unabhängig voneinander programmiert werden können.

Einige Sprachen, etwa COBOL und FORTRAN, sind gemäß allgemeingültiger Richtlinien fest definiert; die Anpassung an neue Hardware- und Verarbeitungstechniken ist dann recht mühsam und langwierig. Dafür erfordert das auf einem Rechner erstellte Programm nur eine erneute Compilierung und nur wenige Änderungen, bis es auf dem anderen Computer läuft.

Sprachen wie PASCAL und C haben nur eine Art de facto-Standard, der von den Erfindern festgelegt wurde. Die meisten Versionen dieser Sprachen entsprechen diesem Standard zwar, bei PASCAL sind jedoch Ein- und Ausgabeprozeduren nicht fest definiert; der Anwender hat also freie Bahn für Zusätze und Änderungen. Die entstandenen Programme sind natürlich nicht problemlos übertragbar.

Ein weiterer Faktor ist das Übersetzungsprogramm. Interpreter sind im allgemeinen einfacher zu bedienen und eignen sich besser für die Programmentwicklung, dafür sind sie aber auch langsamer beim Programmablauf. Compiler dagegen sind schwierig anzuwenden, das Endprodukt arbeitet dafür effektiver. Die Unterschiede nehmen allerdings durch die Entwicklung von Debugger-Programmen und komfortablen Compilern zunehmend ab.

Einige Sprachen arbeiten durch spezielle Hardware-Neuerungen sehr viel schneller. So wurde viel Aufwand in die Entwicklung eines Prozessors gesteckt, der direkt in FORTH arbeiten kann.

Der Hauptfaktor bleibt aber immer die Anwendung, für die eine Computersprache eingesetzt werden soll. Wir möchten Ihnen hier die wichtigsten Sprachen mit ihren Vorzügen und Schwachpunkten vorstellen.

der auszubauen. Gute Strukturierung.
Nachteile: Das Niveau ist oft zu niedrig. Schwer verständlich. Viele verschiedene Versionen.
Anwendung: Programmierung auf maschinennahem Niveau, speziell bei der Hardwaresteuerung.

Sprache: FORTRAN
Vorteile: Standardisiert. Umfangreiche Software-Bibliothek. Sehr viele Funktionen.
Nachteile: Altmodisch. Unbefriedigende Strukturierung. Programme oft unverständlich. Festgelegte Programmgestaltung.
Anwendung: Allgemeine wissenschaftliche, mathematische Programme und Anwendungen aus dem Produktionsbereich.

Sprache: COBOL
Vorteile: Standardisiert, für viele Rechner. Leicht verständlich. Gutes File-Handling.
Nachteile: Schwer zu erlernen. Compiler sind umfangreich und teuer. Schlecht strukturiert, geringe Zahl von Datentypen.
Anwendung: Kommerzielle Datenverarbeitung.

Sprache: C
Vorteile: Viele Datentypen und Strukturen. Gute modulare Struktur. Einfacher Zugang zur Hardware. Programme sind schnell.
Nachteile: Programme unübersichtlich. Für ernsthafte Anwendungen ist das Niveau zu niedrig. Nicht völlig standardisiert.
Anwendung: Systemsoftware als Ersatz für Assembler. Für besonders schnelle Anwendungen.

Sprache: LISP
Vorteile: Gute Listenverarbeitung, vielseitige mathematische Funktionen. Umfangreiche Software und Hilfsprogramme. Bei größeren Rechnern weit verbreitet.
Nachteile: Schwer erlernbar, schwer anzuwenden.
Anwendung: Künstliche Intelligenz und allgemeine Listenverarbeitung.

Sprache: PROLOG
Vorteile: Gute mathematische Basis. Läßt sich einfach einsetzen. Steht dem menschlichen Denken nahe.
Nachteile: Keine rein relationale Sprache, es gibt verfahrensorientierte Ansätze.
Anwendung: Künstliche Intelligenz, Datenbanksysteme.

Sprache: Assembler
Vorteile: Volle Kontrolle über alle Rechnerkomponenten. Effektivste Programmiersprache.
Nachteile: Keinerlei Norm. Schwer zu erlernen und anzuwenden.
Anwendung: Höchste Effizienz, aber nur, wenn sonst „nichts geht".

Sprachgefühl

In der Tabelle ist eine Reihe von Anwendungen aufgeführt. Die passenden Sprachen stehen nach Eignung geordnet rechts daneben.

Anwendung	Sprachen
Statistische Analysen	FORTRAN, BASIC, PASCAL
Lagerverwaltung	COBOL, PASCAL, BASIC
Robotersteuerung	FORTH, C, Assembler
Adventure-Spiele	PROLOG, C, BASIC
Expertensysteme	LISP, PROLOG, LOGO
Ausbildung	LOGO, PROLOG, PASCAL

Das „As"

Als einziger preisgünstiger Heimcomputer ist der „Jupiter Ace" mit FORTH anstelle von BASIC als Standardsprache ausgestattet – eine Herausforderung für alle angehenden FORTH-Programmierer.

Der Jupiter Ace ist insofern ein As, als er zu den wenigen Microcomputern gehört, die nicht von Haus aus mit BASIC arbeiten. Deshalb ist unter seinen Fans immer noch ein schwunghafter Handel mit dem Gerät und der dazugehörenden Software im Gange, obwohl der Rechner seit 1983 nicht mehr gebaut wird.

Die Ausrüstung mit FORTH als Standardsprache hebt den Jupiter Ace deutlich von der Konkurrenz ab. Und er ist gebraucht so billig zu haben, daß Sie damit günstiger wegkommen, als daß Sie sich zum Kennenlernen von FORTH die nötigen Ergänzungen zu Ihrem vorhandenen Rechner kaufen.

Programmieren in FORTH

BASIC-Version

```
100 REM A BASIC PROGRAM TO PRINT 'SHAZAM!'
110 FOR X = 1 TO 6
120 PRINT "SHAZAM!"
130 NEXT X
140 END
    RUN
```

FORTH-Version

```
( A FORTH PROGRAM TO PRINT 'SHAZAM!')
: SHOUT ." SHAZAM! " ; .
:CHORUS 6 0 DO SHOUT LOOP;
 CHORUS
```

Die beiden vorstehenden Programme leisten genau das gleiche, wobei das BASIC-Programm eher an ein Kochrezept erinnert, während die FORTH-Fassung wie ein Zauberspruch aussieht.

In FORTH können auf der Basis eines gegebenen Grundvorrats an Befehlen („Wörter") jederzeit neue Kommandos definiert werden, um das Vokabular zu ergänzen. Im obigen Beispiel ist SHOUT für das Ausdrucken einer Zeichenkette neu eingeführt worden, und CHORUS ist aus dem zuvor definierten SHOUT und einigen Wörtern zusammengesetzt worden.

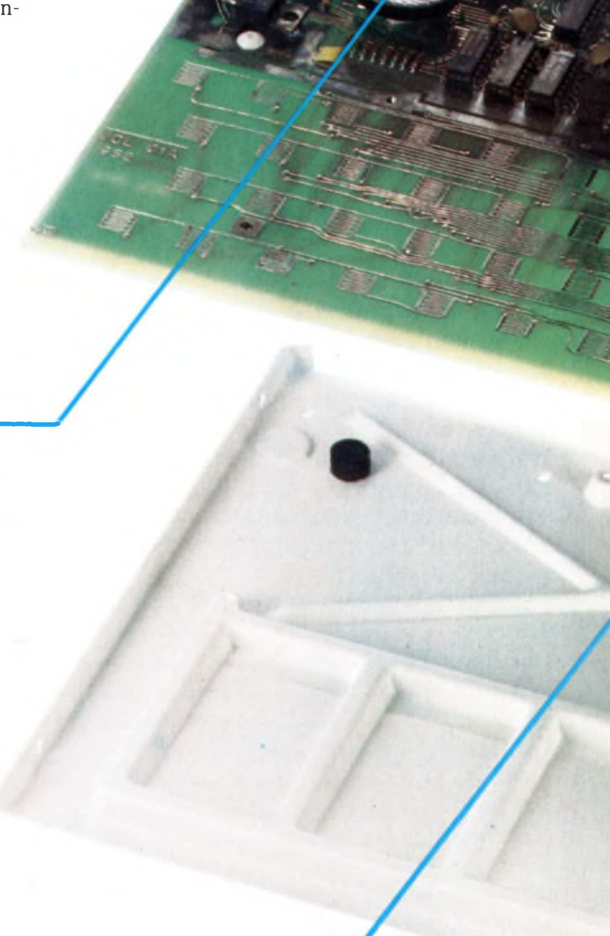
Die Zahlenwertspeicher ist in FORTH als Stapel (Stack) organisiert. Die gewohnten arithmetischen und logischen Operationen sind in FORTH – angepaßt an die Stack-Technik – in „Umgekehrter Polnischer Notation" zu schreiben. Das Programmieren in FORTH erfordert in mehrfacher Hinsicht eine Umstellung, die sich wegen der Leistungsfähigkeit dieser Sprache aber auszahlt. Der Umgang mit FORTH hat einen ähnlichen Reiz wie das Spielen mit dem Rubik-Zauberwürfel, und so etwas macht nun mal dem einen Spaß, während es den anderen langweilt.

Die Vielseitigkeit von FORTH wird dadurch erzeugt, daß neue Befehle definiert werden können. Der Benutzer kann den Wortschatz ganz auf die Bedürfnisse des jeweiligen Anwendungsfalls zuschneiden.



Tastatur
Die Tasten in der obersten Reihe sind alle dreifach belegt. Die Umschaltung erfolgt durch Drücken von SHIFT bzw. SYMBOL SHIFT. Außerdem stehen sieben Grafikzeichen für die Erzeugung einfacher Diagramme und Grafiken zur Verfügung.

Microprozessor
Beim Jupiter Ace wurde als CPU ein Z80 gewählt – das verwundert nicht angesichts der Tatsache, daß hier die gleiche Entwicklungsmannschaft am Werk war wie beim Sinclair Spectrum.



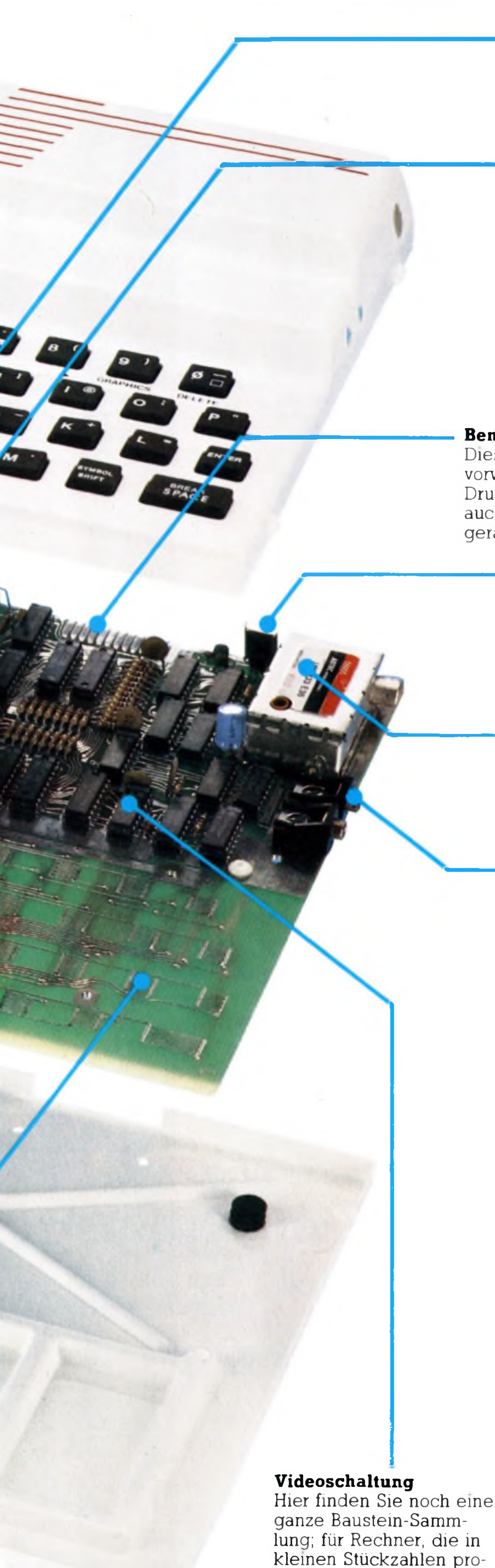


Lautsprecher
Für die Tonwiedergabe sorgt ein Piezoschwinger, wie er in ähnlicher Form auch als Signalgeber bei Digital-Armbanduhren eingesetzt wird. Das genügt für einfache Sound-Effekte.



Kontaktfelder
Beim Betätigen einer Taste wird das Gummi der Tastenträgerplatte gegen das Kontaktbahnenmuster auf der Platine gedrückt und der Stromkreis geschlossen.





Buserweiterungsstecker
Außer für die Speichererweiterung eignet sich diese Steckleiste auch für einen Adapter für das gesamte Sinclair-Zubehör.

FORTH-EPROMs
Das FORTH-System ist in zwei 4-KByte-EPROMs gespeichert. Die Maskenherstellung für ROMs rentiert sich erst bei sehr hohen Stückzahlen, so daß für kleinere Serien nur EPROMs in Frage kommen.

Benutzeranschluß
Dieser Platinenstecker ist vorwiegend für einen Drucker gedacht. Aber auch andere Peripheriegeräte sind möglich.

Taktgeber
Die Z80-CPU hat keinen internen Generator für die Taktfrequenz. Ein separater Quarzkristall erzeugt den 1-MHz-Takt.

HF-Modulator
Mit dem Antennensignal des Jupiter Ace läßt sich ein (Schwarz-weiß-) Fernseher betreiben.

Cassettenrecorder-anschluß

Videoschaltung
Hier finden Sie noch eine ganze Baustein-Sammlung; für Rechner, die in kleinen Stückzahlen produziert werden, ist die Anfertigung besonderer Chips zu teuer.

Äußerlich ist der Rechner eng mit den Sinclair-Modellen verwandt. Wie der ZX80 hat auch der Jupiter Ace ein leichtgewichtiges, weißes Plastikgehäuse. Zur Klangwiedergabe dient ein kleiner Piezokristall, dem sich nach näherer Beschäftigung mit dem Gerät auch etwas mehr als nur einfache Tonfolgen entlocken lassen. Wie die Sinclair-Rechner hat auch der Jupiter Ace einen Transformatoranschluß, eine Antennenbuchse und zwei große mehrpolige Buchsen für Peripheriegeräte. Ein Cassettenrecorder ist über zwei Steckbuchsen anschließbar.

Die Bildschirmausgabe erfolgt schwarzweiß mit 32 Zeichen in 22 Zeilen, im Grafikmode auch 64x48. Jedes Zeichen kann vom Benutzer neu gestaltet werden, um zum Beispiel mathematische Symbole oder Sonderzeichen zu erzeugen.

Der Jupiter Ace ist serienmäßig mit drei KByte ausgestattet. Für kurze FORTH-Programme reicht diese Speicherkapazität jedoch aus. Für höhere Ansprüche gibt es ein 16K- und ein 32K-Erweiterungspaket. Ferner ist ein Adapter erhältlich, der die Verwendung fast der gesamten Sinclair-Peripherie ermöglicht. Außerdem werden ein Centronics-Interface, eine Tonbox und ein Zusatz zur Erleichterung der Tastaturbedienung vertrieben.

Jupiter Ace

ABMESSUNGEN
215 x 190 x 30 mm

GEWICHT
246 g

ZENTRALEINHEIT
Z80A

TAKTFREQUENZ
1 MHz

SPEICHER
3 KByte RAM, extern auf 51K erweiterbar; 8 KByte ROM.

BILDSCHIRMDARSTELLUNG
Schwarzweiß mit 22 Zeilen zu 32 Zeichen, bzw. 64 x 48 bei Grafik.

SCHNITTSTELLEN
Fernseher- und Cassettenrecorderanschluß, 9-V-Versorgungsbuchse; zwei Platinenstecker, wovon der eine alle Daten- und Adreßleitungen des Prozessors führt, der andere nur die Daten- und einige Steuerleitungen.

MITGELIEFERTE SPRACHE
FORTH

MITGELIEFERTES ZUBEHÖR
Steckertransformator, Cassettenrecorder- und TV-Anschlußkabel.

TASTATUR
Gummitasten wie beim Sinclair Spectrum, aber weicher und unpräziser, die genau mittig gedrückt werden müssen; alle mit Wiederholfunktion und Zweifachumschaltung, was die Erzeugung sämtlicher ASCII-Codes ermöglicht.

HANDBUCH
Das Handbuch ist sehr ausführlich und enthält alle wichtigen Informationen. Vom gleichen Verfasser stammen auch die Handbücher für den ZX81 und den Sinclair Spectrum. Auf 180 Seiten wird eine Einführung in FORTH und eine ausführliche Beschreibung des Jupiter Ace gebracht, beides durch zahlreiche Beispiele illustriert.